Universitat de Lleida

TREBALL FINAL DE GRAU

Estudiant: **Jordi Ricard Onrubia Palacios**.

Titulació:

Títol de Treball Final de Grau: Applying Deep-Learning Techniques to Parallel Processes Scheduling within Heterogeneous Distributed Computational Environments

Director/a: Fernando Guirado, Josep Lluis Lerida

Presentació
Mes: Setembre.

Any: 2018.

Abstract

Many companies, organizations and/or universities have accumulated a large number of computing resources grouped in clusters. Cluster Federated Environments arise as a new architecture to take advantage of such resources by joining them and increasing the computing capacity. With this, a new problem appears, a high number of machines and computing resources implies a huge amount of energy consumption.

The scheduling process, responsible for allocating the applications to the system resources, offers the possibility of improve the resource management increasing the system performance and energy efficiency. A considerably amount of research has been done in this field, but given the scheduling problem is classified as an NP problem it is difficult to determine which method is the best one for it.

From the point of view of the artificial intelligence, the advances in hardware and the recent increasing research in this field and specifically in algorithms for deep learning, neural networks have become an essential tool for the research in other scopes such as pattern recognition, system identification or medical diagnosis. Even though some of those problems are too complex to get an optimal solution neural networks have demonstrated give a good enough solutions.

In this Degrees Final Project (DFP) we will try to find if it is possible to use neural networks to find which is the best heuristic to allocate the applications arriving to the system, obtaining the best results in terms of time and energy consumption.

To build the neural network we will make use of an open source library named Tensorflow developed by Google Brain Team and to execute the workloads with a variety of different techniques we will use ClusterGridSim, a Java application that simulates the allocation using the desired technique.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer science has grown notably the last years, and with this the computer systems and the artificial intelligence.

The growth of the computer systems has become a necessity since the requirement for the new applications are higher as the time passes. The solution to this problem started by increasing the power of a single processor, the next step was the super computers, a great step increasing the computer performance as they make us able the usage of a great number of processors for the application execution, the disadvantage int that moment was the great cost in scalability and the fact that the architecture had to be redefined to increase the computer capabilities.

After the 90's, the reduction of the prices in hardware gave us the opportunity to use different resources cooperatively creating a new system architecture called Cluster.

This architecture is composed by many computers connected to take advantage of its computing power and is still in use today due its high scalability at a low price. As nowadays the number of computing resources available is growing exponentially, this encouraged the research and study of developing techniques to take advantage of its full potential. As the concurrent computing demands a high consumptions of energy and time, one of the objectives is to find a method to reduce the energy and time consumption in executing the application without reducing the computational power.

A similar thing happened with the artificial intelligence specifically in the field of the neural networks (NN), since 1986 when David Rumelhart came with the idea of the back propagation, distributing the pattern recognition error through the network, improving the learning of the NNs drastically. With this, in addition to the improvements in the hardware have made possible enormous improvements on them, this improvements had

made possible the usage of the NNs to help research in scientific and technological fields, giving solution to problems that could not be solved before, some of those problems classified by its complexity as NP-Hard or NP-Complete problems.

In order to introduce the scope of this DFP, Section 1.1 introduces the concepts of distributed computing as well as a range of the most important architectures. In Section 1.2 we will introduce the scheduling problem in cluster federated environments as well as important terms such as heterogeneity, workloads, schedule ordering and co allocation. In Section 1.3 we will give a briefly introduction to machine learning, deep learning and the basics of an artificial neural network.

Introduced the basics, in Chapter 2 it is presented the state of art related to Scheduling and Machine learning.

Next, in Chapter 3 we will explain the methodology followed to carry out this DFP.

Finally in Chapter 4 we are going to present the final conclusions and the possible future work based on this project.

1.1 Distributed Computing

A distributed computer system consists of multiple software components that are on multiple interconnected computers, but run as a single system. The computers that are in a distributed system can be physically close together and connected by a local network, or they can be geographically distant and connected by a wide area network. A distributed system can consist of any number of possible configurations, such as mainframes, personal computers, workstations, minicomputers, and so on. The goal of distributed computing is to make such set of computers in the network, work as a single computer [1].

There is a wide range of different architectures to apply in Distributed Computing Systems, even thought we are not working internally with them, we are using them, so we will give a brief explanation about the most related architectures to our work.

- **Cluster Architectures**: In its most basic form, a it is a system comprising two or more computers or systems (called nodes) which work together to execute applications or perform other tasks, so that users who use them have the impression that only a single system responds to them, thus creating an illusion of a single resource. Clusters or combination of clusters are used when content is critical or when services have to be available and / or processed as quickly as possible. Researchers, organizations and companies are using clusters because they need to increase scala-

bility, resource management, availability (understood as the capacity of being available to be used) or processing to supercomputing at an affordable price level [2].

- **Cluster Federation architectures**: They are the conjunction of the different clusters that can be found in the same institution or organization. They were created with the idea of increasing the computational power of one institution by taking advantage of the resources acquired over the years.

- **Grid Architectures**: It is a collection of cluster head nodes used to share the resources across the multiple domains or share resources among many computers to solve large-scale problems. Grid computing enables the use of computer and data resources to solve complex mathematical problems and harnesses a diverse array of machines and other resources to rapidly process and solve problems beyond an organizations available capacity. Grid, is any network of machines, including personal or desktop computers within multiples organizations and it includes computer clusters, clusters of clusters, or special data sources. It is a method of connecting the power of many computers in a network to solve problems requiring a large number of processing cycles and involving huge amount of data, those problems that are beyond the processing limits of individual computers. In Grid, the communication will be held globally [3].

- **Cloud Architectures**: It is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Organized in three different layers:

  IaaS: Infrastructure as a Service, consumer can provision computing resources within provider's infrastructure upon which they can deploy and run arbitrary software, including OS and applications.

  PaaS: Platform as Service, consumer can create custom applications using programming tools supported by the provider and deploy them onto the provider's cloud infrastructure.

  SaaS: Software as ServiceConsumer uses provider's applications running on provider's cloud infrastructure[4].

The chosen architecture for this DFP is the cluster federation architectures.

1.2 Scheduling in Cluster Federated Environments

The Scheduling problem consists on allocating the applications to the available resources in the system, satisfying one or more performance requirements. Finding a proper allocation of the task can result on an improvement of the system usage and execution time and lower levels of energy consumption. However, the problem lies on its computational complexity classified as the NP-Hard Problem[5].

1.2.1 Heterogeneity

One of the characteristics of Cluster Federated Environments is the capability of using different infrastructures accumulated over the years in an organization. Therefore, it is usual to have a Cluster Federated Environment composed by old and new clusters to increase computational power. With this situation, the system would have a vast number of heterogeneous resources working together, with different computational power and energy consumption values. Here is where scheduling becomes crucial, so we can find a good compromise between the execution time and energy consumption[6].

1.2.2 Workloads

The set of applications, also referred to as jobs, that the system has to execute are called Workloads. The characteristics of these workloads are also an important aspect to take into account. We can differentiate the workloads that are executed in the system in two main groups, based on the availability of the jobs at the starting point of the execution. Thus, the workload can be classified as batch or on-line. Below, the main differences between them are presented[6].

- In batch mode, all the applications are already in the system queue at the start of the execution, allowing the scheduler to analyse the characteristics of the workload and prepare the scheduling in advance.
- In on-line mode, the applications reach the system unpredictably during the course of the execution, providing an unknown factor to the scheduler that prevents any future planning.

On this DFP we use workloads in batch mode, as we need to identify the corresponding characteristics of the workload to be scheduled as the input of the neural network.

1.2.3 Scheduling Ordering

To get the best result in measure of time execution and energy consumption it is important to apply an efficient technique, for this there are two types of schedulers[7][8]:
Time Sharing: Time sharing techniques are employed to ensure that the time on a processor is divided into many discrete intervals or time slots which are assigned to unique jobs.

- Local scheduling: the scheduler shares a global run queue. The threads that need to be executed are placed in a queue. As soon as the processor gets free, it removes the next thread from the queue, executes it for certain time and then is returned to the back of the queue. It is easy to ensure the equal share of the machine, but it does not perform well with fine grained communications.
- Gang scheduling: Each job is composed by several threads, good for running on several CPUs at the same time. One of the most important features of gang scheduling is that context switching is coordinated across the nodes. Therefore, all processes can be scheduled and de-scheduled simultaneously. Since the processes of the jobs are scheduled together, gang scheduling has a huge advantage of faster completion time. However, its need of global synchronization overhead in order to coordinate the set of processes is its disadvantage.
- Communication driven co-scheduling: Each node in a cluster have its own scheduler that co-ordinates the communicating processes of parallel job.In order to determine when and which process to schedule, all these algorithms depends on one of the two events i.e. arrival of message or waiting for message.

Space Sharing: the most common scheduler. It provides the requested resources to one particular job until the job is completely executed. The main advantage of space sharing algorithm are low overheads and high parallel efficiencies. Disadvantage of space sharing algorithm is poor average turn around times.

- Batch scheduling: Batch scheduling is employed for the networked set of computers that fit in a single administrative domain. Batch scheduling is most common in managing dedicated clusters for running non interactive job.

Some of the most known types of Batch scheduling are:

- FCFS (First come first serve): Jobs are considered in order of arrival.

- SJF (Shortest job first): Sorts the jobs and executes the shortest first.
- LJF(Longest job first): Sorts the jobs and executes the longest first.
- Advance reservation: Stores resources and produces schedule by using execution time predictions provided by the users.
- Backfilling: The main goal of backfill algorithm is to try and fit small jobs into scheduling gaps. It improvises the system utilization by running low priority jobs in between the high priority jobs. Runtime estimate of small jobs provided by the users are required by the scheduler in order to use backfill.
- Pre-emptive backfilling: If non of the high priority jobs are in the queue, then the resources that are reserved for them can be used for running low priority jobs. But, if the high priority job is received then those resources can be reclaimed for high priority jobs by pre-empting the lower priority jobs.

In this DFP we only treat the scheduling in the space sharing field.

1.2.4 Co allocation

This technique consists of allocating a parallel job by distributing its tasks between two or more different clusters. Thus, the system is able to run applications that require more resources than those available in a single cluster, increasing the applications that can be executed simultaneously. This technique can be used in Cluster Federated Environments thanks to the dedicated nature of the communication links that interconnect the system clusters, as the communication volume and cost of the different tasks can be predicted and taken into account when doing the scheduling.

Taking advantage of the dedicated link that connects the different clusters in a Cluster Federated Environment and that allows the job tasks that are on different clusters to communicate, the co-allocation technique can be used to increase the performance of the system, joining the free resource of each cluster.

Due to the use of co-allocation appears internal fragmentation, defined as the number of free resources in a system that remain free because they cannot be used to allocate a job, as there are not enough resources for the queue jobs to be allocated. And so, with this allocation, the internal fragmentation of the system is reduced and the utilization increased[9].

1.2.5 Performance and Energy Consumption

In order to predict the behaviour of the Cluster Federated Environments when executing a set of jobs, it is necessary to model the environment. The Distributed Computing Research Group of the University of Lleida, has developed some models for this purpose over the last years [10], [11], taking into account not only the performance of the communication links that interconnect the different administrative domains but also the effects on the co-allocation technique. Furthermore, they have presented a new consumption model that creates synergies with the execution model of the group in order to allow the optimization of this metric.

1.2.5.1 Execution Model

In order to estimate the execution time for a job in a heterogeneous Cluster Federated Environment, based on the model presented in [10], we characterize every job by two factors: the Processing Slowdown (PS) and the Communication Slowdown (CS). The PS for job J_j is obtained from the slowest processing node J_j is assigned to, i.e. the allocated node providing the maximum processing slowdown.

$$PS_j = \underset{\forall r: J_j \in N_r}{max} \{PS_j^r\},\ J_j \in \mathcal{J}$$

where PS_j^r is the slowdown of job J_j in node N_r, which is inversely proportional to the node computation power ($1/\Gamma_r$).

The co-allocation of a parallel job J_j consumes a certain amount of bandwidth in each inter-cluster link L_k, which is calculated with the following equation:

$$B_j^k = (t_j^k \cdot B_j) \cdot \left(\frac{\tau_j - t_j^k}{\tau_j - 1} \right),\ J_j \in \mathcal{J}, \mathcal{C}_k \in \mathcal{C}$$

where B_j is the required per-task bandwidth in cluster k, τ_j is the number of tasks in job J_j , and t_j^k is the number of tasks of job J_j allocated to cluster C_k. The first term in the equation is the total bandwidth consumed by tasks of job J_j in cluster C_k, and the second term is the percentage of communication with other clusters. Saturation occurs when co-allocated jobs use more bandwidth than that available, and jobs sharing the link are penalized by an increment in their communication time.

The inter-cluster Saturation Degree SD^k relates the maximum bandwidth B^k of each link L_k to the bandwidth requirements of the allocated parallel jobs, described in the next

equation:

$$SD^k = \frac{B^k}{\sum_{\forall J_j \in N_k} \left(B_j^k\right)}, \ L_k \in \mathcal{L}$$

where B_j^k is the bandwidth of job J_j when it is in node N_k.

When $SD^k < 1$ the link L_k is saturated, and jobs using the link are delayed; otherwise, it is not. Then, the communication slowdown for job J_j and link L_k, which depends on the saturation, is expressed by the following equation:

$$CS_j^k = \begin{cases} \left(SD^k\right)^{-1} & \text{when} SD^k < 1 \\ 1 & \text{otherwise} \end{cases} \quad J_j \in \mathcal{J}, C_k \in \mathcal{C}$$

The communication slowdown for job J_j is the CS_j^k from the most saturated link used, expressed by the next equation:

$$CS_j = max \underset{\forall N_k : J_j \in N_k}{CS_j^k}, \ J_j \in \mathcal{J}$$

Finally, the estimated execution time for a parallel job J_j is calculated by the equation below:

$$T_j^e = Tb_j \cdot tc_j, J_j \in \mathcal{J}$$

where Tb_j is the base time of job J_j in dedicated resources, and tc_j is the time cost factor when a job is allocated. It is assumed that the base-time Tb_j is known from user-supplied information, experimental data, job profiling, etc. The time cost is modelled based on the heterogeneity of the processing resources selected and the availability of the inter-cluster links used. The time cost for job J_j is expressed by the next equation:

$$tc_j = \sigma_j \cdot PS_j + (1 - \sigma_j) \cdot CS_j, J_j \in \mathcal{J}$$

where PS_j denotes the maximum processing slowdown from the allocated resources, CS_j is the communication slowdown from the inter-cluster links, and σ_j is the portion of the total execution time spent on processing [12].

1.2.5.2 Energy Model

The energy model considers that the computing nodes can have two different states, depending on whether they are executing a task (Computing state) or they are idle (Idle state), with no task assigned to them. Hypothesis previously stated by the model expressed by Orgerie in [13]. Once the energy consumed by each node has been defined,

the energy consumption during the execution is estimated by multiplying each consumption value by the time spent in each state. Model defined in the following equation:

$$energy = \sum_{n} (C_n * CT_n + I_n * IT_n), \quad \forall n \in N$$

where Cn is the energy consumed by node n when it is computing and In when it is idle, CT_n is the computing time of the node n and IT_n is its idle time [12].

1.3 Machine Learning

Machine learning is the design and study of software artefacts that use past experience to make future decisions; it is the study of programs that learn from data. The fundamental goal of machine learning is to generalize, or to induce an unknown rule from examples of the rule's application. The canonical example of machine learning is spam filtering. By observing thousands of emails that have been previously labelled as either spam or ham, spam filters learn to classify new messages.

A good definition given by Tom Mitchell [14] is: A program can be said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. And by this definition we can classify this experiences in supervised, unsupervised and semi-supervised learning, which are briefly explained hereafter.

Supervised Learning: A program predicts an output for an input by learning from pairs of labelled inputs and outputs; that is, the program learns from examples of the right answers. Two of the most commons tasks for supervised learning are.

- **Classification**: The program must learn to predict discrete values for the response variables from one or more explanatory variables. That is, the program must predict the most probable category, class, or label for new observations.
- **Regression.**: The program must predict the value of a continuous response variable.

To make possible the learning we have three different kinds of data sets.

- **Training dataset**: A data set containing training examples with associated correct labels. This will make possible identify patterns from the labelled examples so it can later classify future data.

- **Test dataset**: Independent of the training dataset, but that follows the same probability distribution as the training dataset. A test set is therefore a set of examples used only to assess the performance.

- **Validation dataset**: A validation dataset is a set of examples used to tune the hyper-parameters of a classifier. It is sometimes also called the development set. In artificial neural networks (ANN), a hyper-parameter is, for example, the number of hidden units.

Unsupervised Learning: Unlike supervised learning, unsupervised learning find patterns where we do not. This may be because the "right answer" are unobservable, or infeasible to obtain, or maybe for a given problem, there is not even a "right answer" per se. The most common tasks for unsupervised learning are.

- **Clustering**: Assigns observations to groups such that observations within groups are more similar to each other based on some similarity measure than they are to observations in other groups.

- **Dimensionality reduction**: Is the process of discovering the explanatory variables that account for the greatest changes in the response variable.

Semi-Supervised Learning: This problems make use of both supervised and unsupervised data. An example of semi-supervised machine learning is reinforcement learning, in which a program receives feedback for its decisions, but the feedback may not be associated with a single decision. An example would be a program that learns to play a video game such as Super Mario Bros. It would receive a reward when it completes a level and a punishment when it loses a life. However, this supervised feedback is not associated with specific decisions to run like avoiding enemies or picking coins and other items[15].

1.3.1 Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound.

Computer models are trained by using a large set of labelled data and neural network architectures that contain many layers.

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks, those containing a large amount of hidden layers.

One of the most popular types of deep neural networks is known as convolutional neural networks (CNN). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images. This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

CNNs learn to detect different features of an image using tens or hundreds of hidden layers. Every hidden layer increases the complexity of the learned image features [16][17].

1.3.2 Artificial Neural Network

Artificial Neural Networks is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. An ANN is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process[18]. The main advantages are:

1. Ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques.

2. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.

3. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.

4. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.

5. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

To a better understanding about what is a NN is important to know the differences between neural networks and conventional techniques.

Conventional techniques use an algorithmic approach i.e. following a set of heuristics to solve a problem, for this it is required to have the knowledge to solve the problem. Neural Networks work differently, they learn by example. Thus, the examples must be selected carefully otherwise the network might not work properly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

Neural networks and conventional techniques are not in competition but complement each other. There are tasks are more suited to an algorithmic approach like arithmetic operations and tasks that are more suited to neural networks, normally, conventional techniques are used to supervise the neural network in order to perform at maximum efficiency.

1.3.2.1 Arquitecture of neural Networks

- A Neuron: A neuron represented in the Figure 1.1 is a computational unit that consist in an input set, an output set, a connection between the neuron and the inputs, an activation function and weights for each connection[19]. In a neuron the value

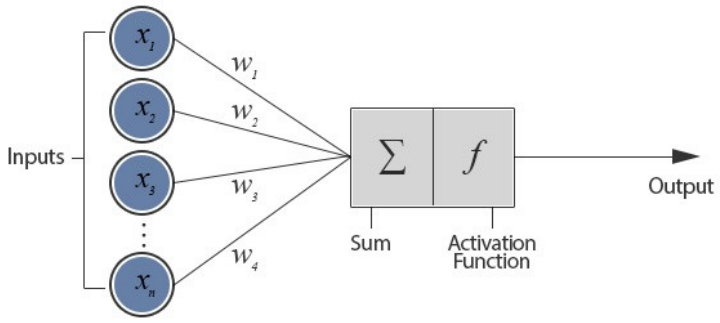

Figure 1.1: Representation of a neuron[20].

of each input is multiplied by the weight of the connection. The addition of all the

input multiplied is the input to the neuron, then, the neuron does its operation and produces the input.

The most used model is the one based on *threshold logic*, where to consider the output active, we compare the addition of the inputs multiplied by its weights with a prefixed value, if the prefixed value is exceeded the output is active.

Therefore, given a neuron with n inputs, represented by the input vector $(x1, x2, ..., xn)$, a set of weights $(w1, w2, ..., wn)$ and a prefixed value T, the neuron will be activated if the inputs fulfil the equation:

$$w1xi + w2x2 + ... + wnxn > T$$

This equation can be remodelled to simplify the learning algorithms for NN, for this we will use the prefixed value T as a weight, we are going to introduce a new input with the value -1 and the weight T. Now the input vector and weight vector are $x = (x1, x2, ..., xn, -1)$ and $(w1, w2, ..., wn, T)$, Then the equation becomes:

$$w1xi + w2x2 + ... + wnxn + T(-1) > 0$$

Now we have all the values in the equation unified as weights.

When we consider a neuron as a system of automatic classification, usually, the input vector is associated with a particular object that we want to classify given certain class of objects. If the neuron is active, the object is recognized as a part of that class, otherwise is not. Then we will say that a neuron gives a proper classification if the neuron is active for the objects belonging to the class.

- A Neural Network: One neuron is enough for representing simple cases, e.g. an AND or OR logic gate, as we can see in the Figure 1.2 we only have to differentiate between the elements up and down the cut.

  For a more complex case as in a XOR logic gate, we have another cut, as we cannot differentiate the points with only one cut, a single neuron is not able to resolve this problem, and for this, it is not able to resolve more complex problems were the functions are represented with more points. Neural Networks are modelled as collections of neurons that are connected in an acyclic graph. In other words, the outputs of some neurons can become inputs to other neurons. Neural Network models are often organized into distinct layers of neurons. For regular neural networks, the most common layer type is the fully-connected layer in which neurons between two

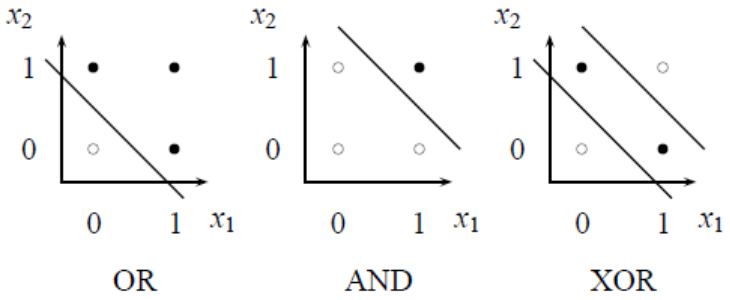

Figure 1.2: Representation of an OR, AND an XOR functions, the cuts represents the separation for the valid values(black dots)[21].

adjacent layers are fully pairwise connected, but neurons within a single layer share no connections[22][23][24]. As we can see in the Figure 1.3 a neural network has at

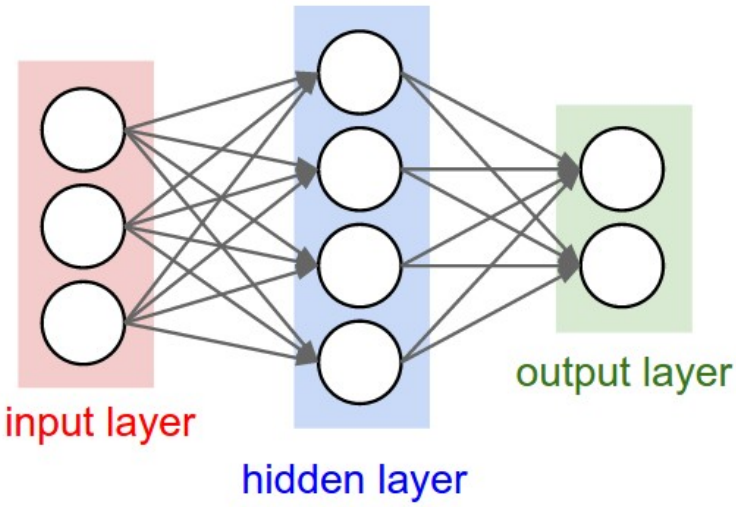

Figure 1.3: Representation of a Neural Network with an input layer, a hidden layer and an output layer [25].

least three layers, the first one the input layer, where the input neurons are placed, in this layers the neurons will get the input from the outside of the neural network. The most right layer is called the output layer, the output neurons will be in charge to give the final result. The middle layer is called hidden layer the neuron in this layer are not inputs nor outputs, here is where the actual processing is done and the

final result will be sent to the output layer. There is no actual method to know how many hidden layers are needed for a neural network. Instead, neural networks researchers have developed many design heuristics for the hidden layers, which help people get the behaviour they want out of their nets.

- Back-Propagation Algorithm: The most used algorithm for learning is the back-propagation algorithm. Back-propagation is about understanding how changing the weights and biases in a network changes the cost function. For a neural network with n layers, and with n as the output layer, we define the error functions associated to the training set as:

$$E = \sum_s E_s = \sum_s \sum_j 1/2(y_j^s - f_j^n)^2$$

where y_j^s denote the desired value for the neuron j at the output layer when we feed the network with the sample s and f_j^n the value that returns the neuron. Given the consideration at the error function that the training samples are fixed values, then the E function is a function that changes when we change the weights of the network, since the value of the neurons in a layer depends on the value of the neurons at the previous layer, any global change of the weights can affect at the value of the neurons at the output, and therefore, at the error function. We are interested in reach a global minimum at the function but generally, it is not possible, therefore, what we are going to do is try to reach a local minimum[22][26].

The idea behind the back-propagation algorithm is an implementation of the gradient descent applied to the error function. With this, what we try to do is reach the local minimum at enough speed so we can find it fast, but without going so fast that we passed by it. The inconvenience is, if we have more than one minimum local, the algorithm can modify the weights so it will be moving between the locals minimums but never finding the global minimum. What we want to detect is, given a weight, for which variation of its value it produces a descend of the error function and how fast it is. For this, we will use the partial derivative of the error function.

The error functions is the addition of the errors given by all the samples, the final gradient is obtained by the addition of each gradient associated at the error given by a sample s. The error produces by this sample s would be:

$$\nabla E_s = (\frac{\partial E_s}{\partial w_{1,1}}, \frac{\partial E_s}{\partial w_{1,2}}, ..., \frac{\partial E_s}{\partial w_{n,m_n}}) \tag{1.1}$$

It has as many terms as different weights has the neural network. The objectives, then, is calculate all the partial derivatives. Each one, indicate us how much and in what direction the error function changes when we increment the weight without modifying the others. Given a weight wi, j that connects the output of the neuron i and the input of the neuron j, when we have calculated the value for all the samples, the change will be proportional to the addition of all the partial derivatives:

$$w_{i,j} = w_{i,j} - r \sum_{s} \frac{\partial E_s}{\partial w_{i,j}} \tag{1.2}$$

Where r is the learning rate. The bigger the learning rate the faster we move to the local minimum i a single move. What the back-propagation algorithm does is modify all the weights using the previous equation, with that we will reach another point in the error function. If the error is not low enough, the gradient is again calculated, and the weights are modified again, this process is repeated until we reach an enough low error level.

Chapter 2

State of art

In this chapter we will cover the most recent situation about the main concepts and the works present in the lifetime related to this DFP, Scheduling and Machine Learning.

2.1 Scheduling

Scheduling techniques have evolved together with computing systems architectures.Initial studies about distributed systems were undertaken by Feitelson and Rudolph in [27]. Since then, many improvements have been developed that improved the architecture of these systems [28].

More modern research studied the evaluation of all the jobs in the queue, treating them together, in order to find the optimal solution. Blanco et al. [29] [30] proposed diverse techniques to determine the best scheduling of sets of job packages, proposing a new job execution order to minimize their overall execution time based on a Mixed-Integer programming model. However, finding an optimal solution to the scheduling problems using this technique has a high computational cost due to the large amount of data to process. For this reason, nature-inspired meta-heuristics, such as Simulated Annealing (SA), Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), have emerged as effective techniques in complex large-scale environments in an attempt to obtain pseudo-optimal solutions in practical times. GAs are well known for their robustness and have been applied successfully to solve scheduling problems in a variety of fields. Zomaya and Teh [31] used GAs in dynamic load balancing problems. Braun et al. [32] compared the efficiency of a simple GA-based scheduler and the well-known techniques such as MinMin, MinMax and Minimum Completion Time (MTC) algorithms. Carretero

and Xhafa presented, in [33], an extensive study of GAs to design efficient Grid schedulers where makespan and flowtime are minimized to include QoS in the solutions, but considering independent jobs without inter-cluster communications. PSO is also utilized in scheduling techniques, as it provides similar results to GA but with a simpler and less costly algorithm, which allows a faster convergence to the near optimal solutions. Thanushkodi, in [34], presented a PSO to schedule tasks in a multiprocessor system in order to reduce the waiting times and execution time of jobs. Surendra, in [35], presented a PSO scheduling technique focused on scheduling jobs in GRID systems.

Reaching the modern era, the society has acquired a better awareness of the environment and ecological sustainability of all technologic progress. Thus, energy consumption has become a great challenge in the field of high-performance computing. In order to reduce energy consumption, two primary methods are commonly used: switching off underutilized resources [36][37][13] and using voltage and frequency scaling (VFS) techniques [38][39][40]. Cocaña et al. [36] presented a software tool that predicts the future node requirements using a machine-learning approach, and then stopping those that will not be required in the near future. Chae et al. [37] illustrate a method to determine the aggregate system load and the minimal set of computational resources that can process the workload. Orgerie et al., in [13], present a three-step strategy based on a framework able to control the computing requirements by switching the unused nodes off, predicting their usage in order to switch them on again and finally aggregating some reservations in order to avoid frequent on/off cycles. Christobel et al., in [38], proposed an energy-aware scheduling approach for scientific workflows based on Particle Swarm Optimization and Dynamic Voltage Scaling[41].

In one of the latest works made by Eloy Gabaldón [42][43][44][45], differents proposals are presented, the first focused on the optimization of one objective while the last one is a multi-objective hybrid method which has been used to take advantage of the strengths of these heuristic algorithms and overcome their shortcomings in terms of dispersion and convergence.

As we can see, there is a wide range of heuristics to use, the problem resides in that currently there is not known method to determine which heuristic will work better for an specific situation.

2.2 Machine Learning

2.2.1 Image Recognition

In the last few years machine learning has made tremendous advancements on image recognition, this progress has been made thanks to the discovery of the deep convolutional neural network(CNN) model, achieving reasonable performance on hard visual recognition tasks, sometimes even exceeding human performance.

This advancements had helped us to create growth opportunities in many fields, one of the most used is facial and fingerprint recognition. Another interesting usage is the detection of possible abnormal cells or tissues in medical image. Furthermore they can be used to create art, neural networks can be trained to learn about the most important details in paintings making it possible for them to recreate new one, as an example we have the "new" Rembrandt created by an artificial intelligence by learning patterns from others paintings of the same author.

2.2.2 Natural Language Processing

Natural Language Processing (NLP) is a tract of Artificial Intelligence and Linguistics, devoted to make computers understand the statements or words written in human languages[46].

- **Machine Translation**: The challenge with machine translation technologies is not directly translating words but keeping the meaning of sentences intact along with grammar and tenses.

  The statistical machine learning gathers as many data as they can find that seems to be parallel between two languages and they crunch their data to find the likelihood that something in Language A corresponds to something in Language B. Google, in September 2016, announced a new machine translation system based on Artificial neural networks and Deep learning .

  In recent years, various methods have been proposed to automatically evaluate machine translation quality by comparing hypothesis translations with reference translations.

  Examples of such methods are word error rate, position-independent word error rate, generation string accuracy, multi-reference word error rate. All these criteria

try to approximate human assessment and often achieve an astonishing degree of correlation to human subjective evaluation of fluency and adequacy.

- **Text Categorization**: Categorization systems inputs a large flow of data like official documents, military casualty reports, market data, newswires etc. and assign them to predefined categories or indices.

  Some companies have been using categorization systems to categorize trouble tickets or complaint requests and routing to the appropriate desks.

  Another application of text categorization is email spam filters. Spam filters is becoming important as the first line of defence against the unwanted emails. A false negative and false positive issues of spam filters are at the heart of NLP technology and Text Categorization, its brought down to the challenge of extracting meaning from strings of text.

- **Information Extraction**: Information extraction is concerned with identifying phrases of interest of textual data. For many applications, extracting entities such as names, places, events, dates, times and prices is a powerful way of summarize the information relevant to a user's needs.

  In the case of a domain specific search engine, the automatic identification of important information can increase accuracy and efficiency of a directed search.

  Discovery of knowledge is becoming important areas of research over the recent years. Knowledge discovery research use a variety of techniques in order to extract useful information from source documents like: Parts of Speech (POS) tagging, Chunking or Shadow Parsing, Stop-words (Keywords that are used and must be removed before processing documents), Stemming (Mapping words to some base for, it has two methods, dictionary based stemming and Porter style stemming.

- **Dialogue System**: The most desirable application of the future.

  Dialogue systems, which focuses on a narrowly defined applications (like refrigerator or home theatre systems) currently uses the phonetic and lexical levels of language.

  These dialogue systems when utilizing all levels of language processing (Phonology, Morphology, Lexical, Syntactic, Semantic, Discourse, Pragmatic) offer potential for fully automated dialogue systems .

Today we can find dialogue systems available for all kind of users, some examples are, Google's assistant, Windows Cortana, Apple's Siri and Amazon's Alexa.

- **Medicine**: The Linguistic String Project-Medical Language Processor is one the large scale projects of NLP in the field of medicine.

  The LSP-MLP helps enabling physicians to extract and summarize information of any signs or symptoms, drug dosage and response data with aim of identifying possible side effects of any medicine while highlighting or flagging data items.

  The National Library of Medicine is developing The Specialist System. It is expected to function as Information Extraction tool for Biomedical Knowledge Bases, particularly Medline abstracts.

We can observe that the usages of the machine learning has expanded to a wide variety of fields, but not the scheduling. In the present DFP, we propose an innovative approach to the scheduling problem by using machine learning capacities to identify the most suitable scheduling method for any workload.

Chapter 3

Methodology

3.1 Objective

The aim of this project is:

The study of the availability of using machine learning to predict beforehand the proper heuristic for the scheduling of an application in a Federated Cluster Environment to get the optimal results in function of makespan (the time elapsed since the first job starts its execution until the completion of the last job) and the energy consumption during the applications execution.

3.2 Research Methodology

The usage of machine learning with the previously mentioned purpose is something with slight presence, therefore there is not knowledge about what is the proper procedure to follow. In this DFP we present the following methodology followed with a visual representation in Figure 3.1 to achieve the objective.

1. **Workload Generation.**
   This parts consists on the generation of an enough amount of workloads, those, based on real workload sets that will be parsed and filtered as its described in Section 3.3 to be executed later.

2. **Workload Execution and Analysis.**
   We execute the workloads generated previously and analyse the obtained results as its described in Section 3.4, with this step we obtain the best policy for each workload to later generate the proper amount of datasets that will be used in the NN. For

the execution we will use GridSim, a Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing[47].

3. **Workload Representation.**
   We need a representation of the workload to be passed to the NN. The NN will use this representations to learn how to distinguish which method is the appropriate for each workload based on it. In Section 3.5 we cover that matter.

4. **Neural Network Design, Execution and Analysis.**
   In Section 3.6 we will cover the NN design, the execution, as well as the size of the sets that we use, and the results obtained for this, the reason behind this three steps are grouped in the same section is because its relation, as the depending on the results we might have to change the design of the NN, tune it in different ways or change the sizes of the tests sets.

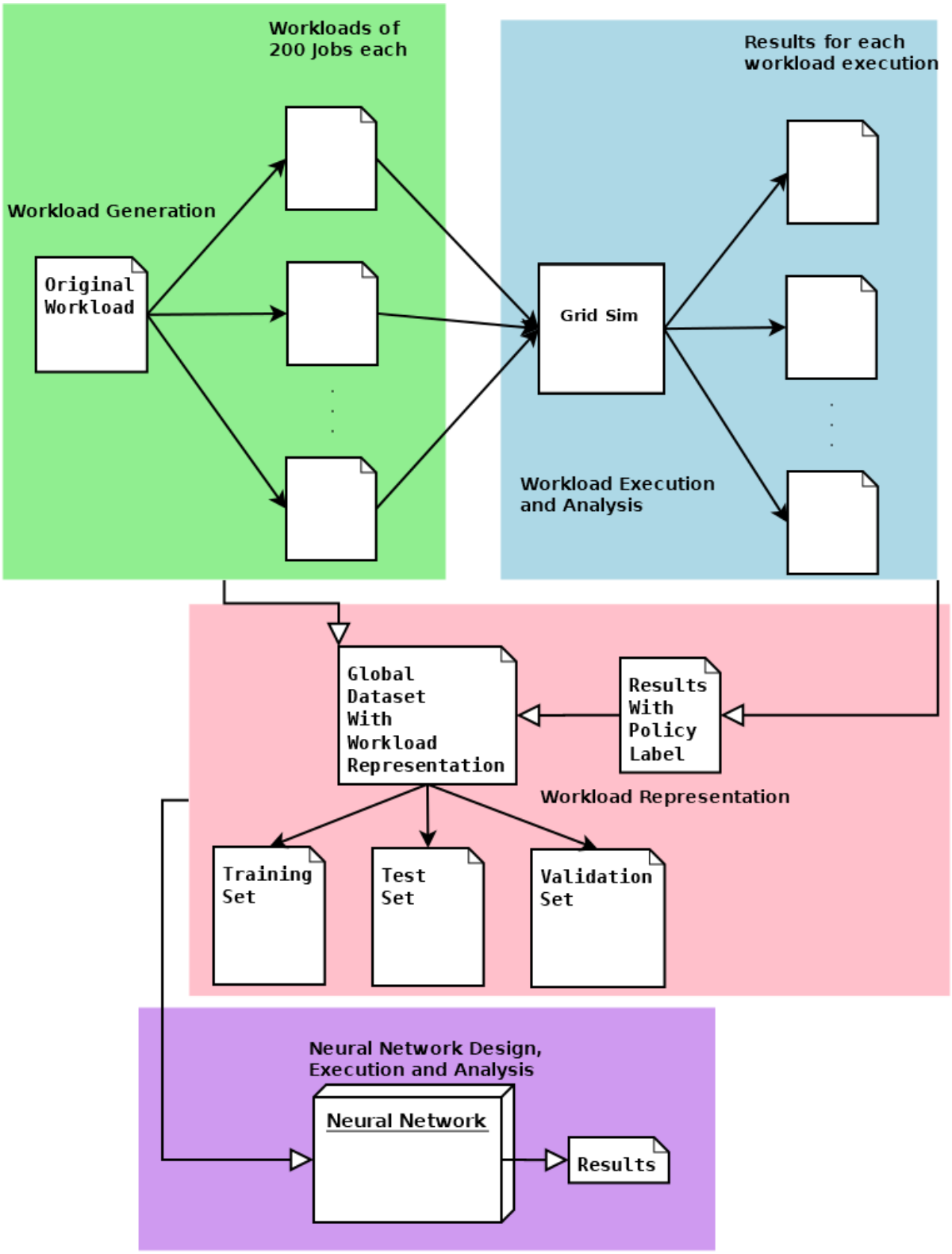

Figure 3.1: Methodology Diagram[48].

3.3 Workload Generation

We need to obtain an amount of workload sets, that will be executed in the GridSim with each policy, to obtain the makespan and the energy for each workload, once we get this values we can find which is the best policy to schedule each workload.

To obtain the workload sets we will use Logs of Real Parallel Workloads from Production Systems[49], specifically we are going to use three different log files, CEA-CURIE, HPC2N and RICC, all of them with different characteristics presented as follows.

- CEA-CURIE: 50% of the jobs made up of 32 tasks. The execution time of 80% of the jobs being extremely low, from 1 second to 1 minute.
- HPC2N: Half of the jobs contain up to 1-2 tasks, the rest are evenly distributed in 4, 8 and 16 tasks. The average execution time is around 1 hour.
- RICC:80% of jobs are made up of 1 task while the rest can be up to 32. The execution times of each job can vary greatly from few seconds to 10 hours.

We can not use the workload directly as its format is not the proper to be executed with the GridSim.

GridSim takes into account 4 different parameters present in Table 3.1 and described below.

- Column number 4 (pink), runtime. It describes the running time of the job, as all of the log files belongs to real parallel workloads, all of them have been executed before, therefore, all of them have runtime and we don't have to apply any condition to this field.
- Column number 5 (orange), used cores. Number of cores used, they represent the tasks of the job. In some cases we can find that there is no information about it, represented as -1, or that the job used more cores than the we have available, 200 cores, then, we have to remove all the jobs that do not stick to our requirements.
- Column number 19 (blue), processing communication proportion or sigma. This column is not in the original log and we have to generate it randomly, the possible values for this column are 0.6 or 0.8, for this we generate a random number , if this random number is a 0 then we will write a 0.6 otherwise we will write a 0.8.

- Column number 20 (purple), last value. As the sigma column this one is neither in the original log and we will have to generate it randomly, the possible values for this column are between 0.0 and 0.5, for this we will use the method random.uniform(0.0, 0.5) which will give us a number between the desired values.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
8	0	208	5	128	-1	-1	-1	12	-1	1	4	71	-1	-1	4	-1	-1	0.8	0.482
9	0	134	1	64	-1	-1	-1	5	-1	1	7	13	-1	-1	4	-1	-1	0.8	0.352
10	0	111	5	32	-1	-1	-1	11	-1	1	4	97	-1	-1	4	-1	-1	0.8	0.163
11	0	154	3	64	-1	-1	-1	65	-1	1	4	28	-1	-1	4	-1	-1	0.6	0.245
12	0	675	3	64	-1	-1	-1	28	-1	1	2	71	-1	-1	4	-1	-1	0.8	0.451
13	0	432	9	16	-1	-1	-1	19	-1	1	4	54	-1	-1	4	-1	-1	0.6	0.453

Table 3.1: Workload Line Example

Once all the jobs inside the log meet our requirements, we will divide the workload in pieces of 200 jobs each, so we can generate an enough amount of sets, the new workloads will have jobs from the bigger workload picked randomly and without repetition, this process will be repeated until we have the desired amount of workloads.

3.4 Workload Execution and Analysis

Once we have the workloads with a proper format we will execute them one by one with each possible policy, then we will proceed to look for the best values in terms of makespan and energy.

The chosen policies are well known techniques that have been previously tested in GridSim, described below followed by its characteristics, Table 3.4, the policies have been distributed taking into account if they need parameters for its execution or not [50].

Without parameters:

- FCFS is a classic technique that allocates tasks of the first job in the queue to the available nodes.
- CBS was proposed by Jones et al. [51] agglomerates tasks from the job in a single cluster in an attempt to avoid inter-cluster link saturation.

- JPR is a variant of the technique proposed by Naik et al. [52], where jobs are allocated to the best available resources depending on the criterion selected to minimize: makespan or energy.
- Min-Min presented by Li et al. [53] minimizes the energy consumption during the scheduling process.

With parameters:

- HILL is a local search method based on the hill-climbing technique, focused on minimizing a fitness function defined by a weighted value of energy and makespan: $\alpha \times makespan + (1 - \alpha) \times energy$.

  The parameters used for its execution are presented in the Table 3.2.

Package Size	Number of Restarts	Neighbourhood Size
200	100	100

Table 3.2: Hill Parameters

- MOGA is a multi-objective genetic algorithm in [54] for minimizing both makespan and energy consumption.

  The parameters used for its execution are presented in the Table 3.3

Package Size	New Generations	Population Size	Crossover	Mutation
200	60	80	80	10

Table 3.3: Moga Parameters

Technique	Modifies order	Mapping criteria	Iterative
FCFS	No	Select available nodes	No
CBS	No	Reduce inter-cluster link saturation	No
JPR	No	Energy or Makespan	No
Min-Min	Yes	Energy	No
HILL	Yes	Energy and Makespan	Yes
MOGA	Yes	Energy and Makespan	Yes

Table 3.4: Policies Characteristics

Once the execution with all the policies has finished for each workload, we will keep the workload labelled with the policy that has produced the best result. Once all the workloads are labelled we will analyse the percentage of workload for each policy. The policies that have 0% will be removed as there is no point on classify something that does not exist.

For the results with low percentage of presence after the execution it will be necessary to analyse the values of makespan and energy, if the values of the best policy is similar to other policies that have more representation it can be worth changing the label for the other policy with higher representation.

3.5 Workload Representation

For the workload representation we will use the values that are used for the gridSim application to calculate the makespan and the energy, the explanation about the treatment of each values as well as a graphic example 3.8 can be found hereunder.

- Task: We know that the minimum of tasks a job is going to have is 1 and also the maximum is limited to 200, with this information the first group of numbers will be how many task per job has each workload, having 10 groups where the first groups those with 1 - 20, the second 21 - 40, and so on.

- Runtime: As it is said in Section 3.4 each log file has different characteristics on the runtime, those represented in Figures 3.2 3.3 3.4 all of them have a bigger quantity of short workloads than long, therefore, there is a higher chance that in the smaller workloads the proportion of the jobs with short runtime happens to be bigger than

the jobs with longer. Also, there range between the lowest value and the greatest is considerable, for this we have decided to group the runtime, with this, we avoid having a line with all the possible values inside the workload and additionally, in reference to the greater amount of lower values, discretizing this values we obtain a clearer representation making the NN learn about the weight on each group, if we did not group them we would have a humongous number of values to 0, one for each value not present in a workload, that would make the learning difficultier as they would not mean anything, in other words a 0 is telling us that there is no value while we are more interested in the values that are present.

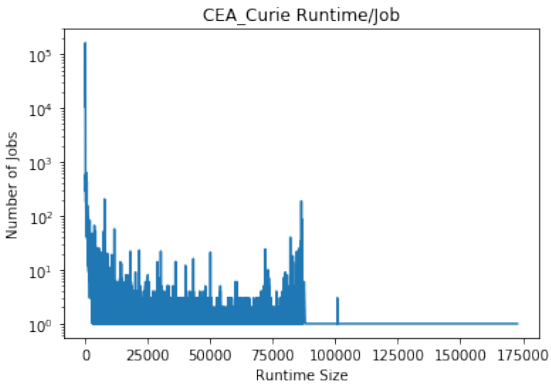

Figure 3.2: CEA-curie Runtime/job distribution[55].

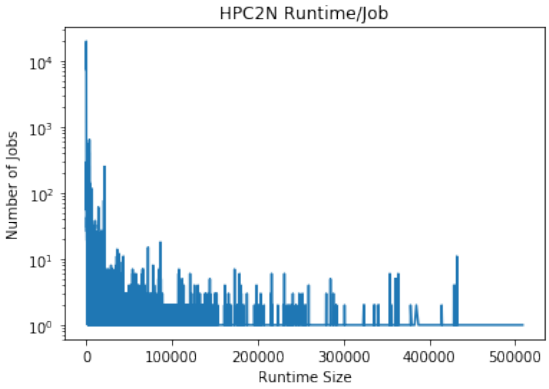

Figure 3.3: HPC2N Runtime/job distribution[56].

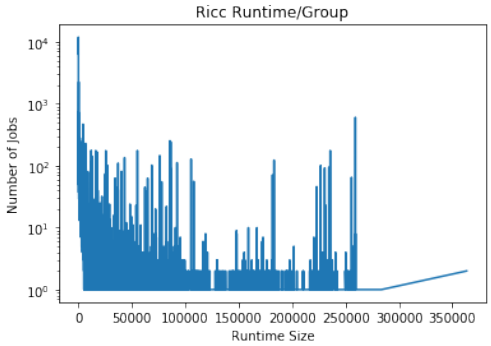

Figure 3.4: RICC Runtime/job distribution[57].

To avoid having useless data we have to discretize it. We will try to distribute de data so the difference between the number of values of each group differ the minimum.

The ranges for each log, obtained in a experimental form by making first 5 groups with equal number of values in each range, then, reducing the bigger groups observing the Figures 3.2 3.3 3.4 to make them more discrete, those ranges are presented as follows:

- CEA-Curie $[1-4]$, $[5]$, $[6]$, $(6-250]$, $(250-..]$
- HPC2N $[1-11]$, $(11-300]$, $(300-3000]$, $(3000-6000]$, $(6000-..]$
- RICC $[1-40]$, $(40-400]$, $(400-2400]$, $(2400-8000]$, $(8000-..]$

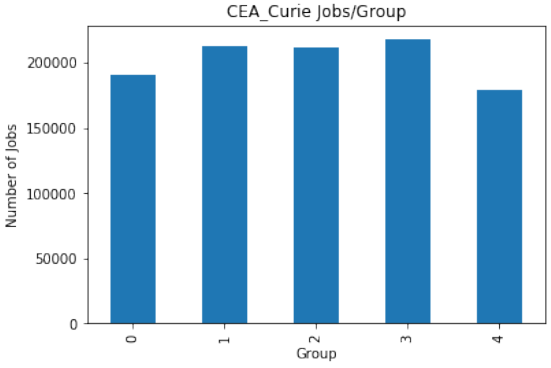

Figure 3.5: CEA-curie Job distribution/Group[58].

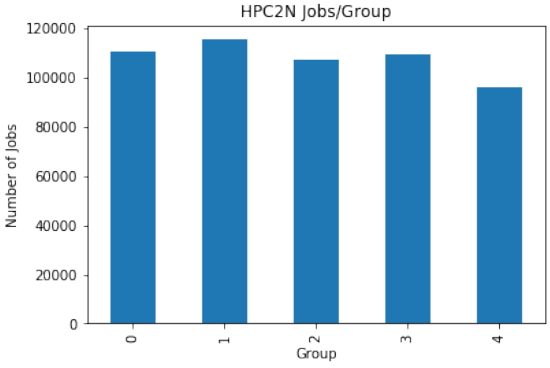

Figure 3.6: HPC2N Job distribution/Group[59].

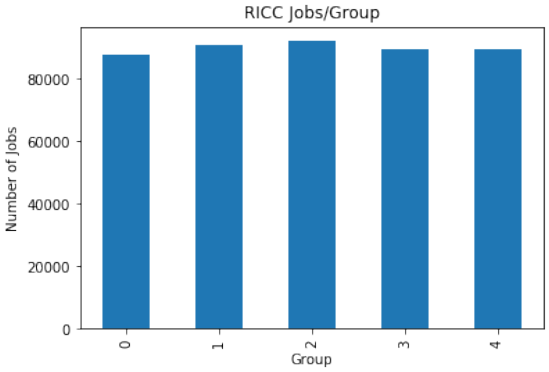

Figure 3.7: RICC Job distribution/Group[60].

As we can see in Figures 3.5 3.6 3.7, the runtime groups have a better distribution, this should reduce the number of 0 values in the workloads making the learning easier.

- Computing/Communication Distribution: The computing/communication distribution only can have two possible values, 0.6 and 0.8, we group the jobs by those values.
- Last Value: The possible values of this field is compressed between 0.0 and 0.5, so we will have 5 range of numbers $[0.0 - 0.1)$, $[0.1 - 0.2)$, and so on.

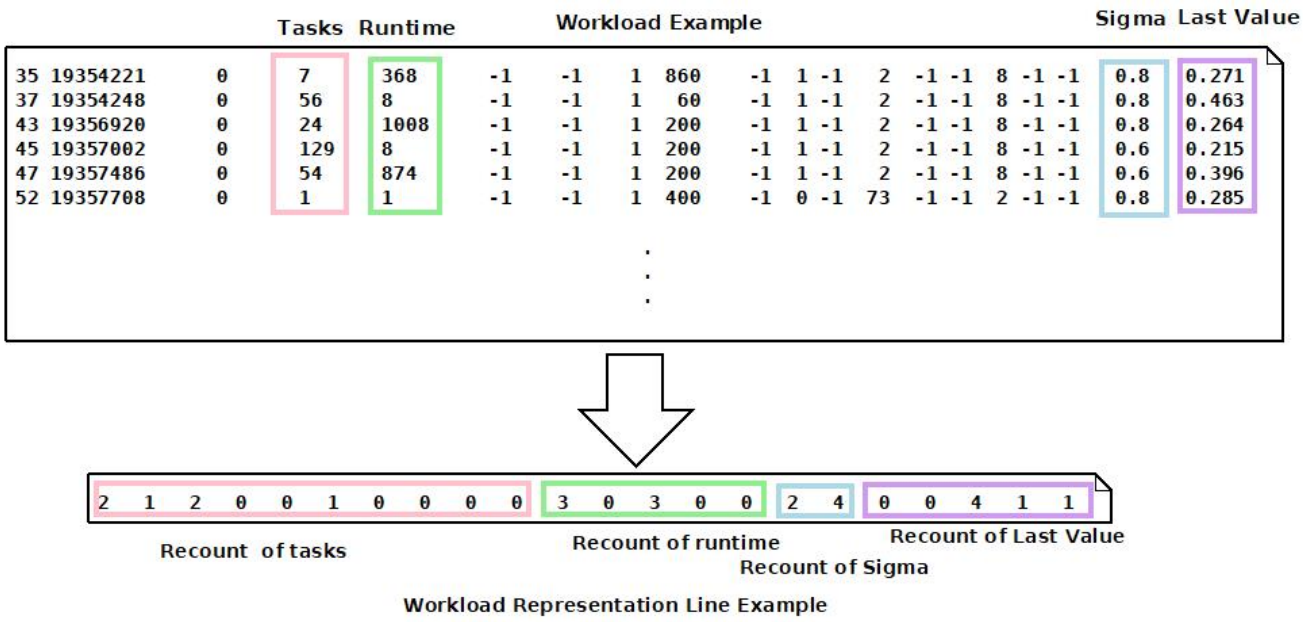

Figure 3.8: Representation of a Single workload[61].

3.6 Neural Network Design, Execution and Analysis

To build the NN we will use TensorFlow. Tensorflow is the second machine learning framework that Google created and used to design, build, and train deep learning models[62].

It defines computations as a data flow graphs. In these graphs, nodes represent mathematical operations, while the edges represent the data, which usually are multidimensional data arrays or tensors, that are communicated between these edges.

In a data flow graph each node, each operation will have as many inputs as needed and as many outputs as needed including 0.

3.6.1 Training Set and Validation Set

For the creation of the training sets and test sets we need to take into account the number of results obtained in the execution of the GridSim, as our objective is to find the best method, we are going to filter the results to obtain the best method for each workload then assign the method to its representation. The workloads have been obtained from 3 different real sets of workloads, each set with the number of belonging workloads are present on the following Table 3.5.

Workload Name	Quantity	Percentage
CEA-CURIE	58573	35.53 %
HPC2N	36261	22.00 %
RICC	70000	42.47 %
Total	164834	100%

Table 3.5: Used Workloads

The classification method/quantity of workloads from the total seen in the previous Table 3.5, can be seen in the next Table 3.6.

Method	Quantity	Percentage
FCFS	58	0.0351 %
CBS	55	0.0334 %
JPR	277	0.1681 %
Min-Min	2	0.0012 %
Hill-Climbing	160051	97.0983 %
MOGA	4391	2.6639 %

Table 3.6: Results Classification

It is easily observable that Min-Min will not be part of the data sets, 2 results are not enough to make a single training set nor validation set. The distribution in terms of training and validation set for the rest of the methods can be seen on the following Table 3.7.

Number of Methods	Methods	Training Set Size	Validation Set Size
2	Hill-Climbing, MOGA	3200	800
2	Hill-Climbing, MOGA	1200	300
2	Hill-Climbing, MOGA	560	140
3	JPR, Hill-Climbing, MOGA	150	40
3	JPR, Hill-Climbing, MOGA	100	30
3	JPR, Hill-Climbing, MOGA	50	30
5	FCFS, CBS, JPR, Hill-Climbing, MOGA	50	5

Table 3.7: Data Sets Distribution

With that distribution we try to make a wide variety of sets with different sizes with the intention of avoiding overfitting, and as we do not know which is the best size for the sets and how hard it is learning something, we have included smaller distributions like the one formed for JPR, Hill-Climbing and MOGA with 50 training sets and 30 validation sets and the one formed for FCFS, CBS, JBS, JPR, Hill-Climbing and MOGA with 50 training sets and 5 validation sets, both occupying the two lasts positions of the Table 3.7.

An important detail is that there is no repeated or shared value in the same training or validation set. This decision is logic as learning from same values only will cause the NN to learn an specific pattern, the same for the validation, it is a lot easier for a NN to recognize a previous studied pattern than generalize from the whole set previously seen.

3.6.2 Neural Network Design

The kind of neural network chosen for this paper is called Estimator, an Estimator is a simply high-level Tensorflow API that suits the previously chosen representation for our workloads. The reason behind is that even though we could use an Estimator for matrix representation data sets, those require more aspects to take into account as they hold more information than a vector. Thus, we are not going to overcomplicate which is already complicated.

Now we are going to present the advantages of using an Estimator:

- Generally easier to create models.
- Build the graph "automatically", you still have to decide the configuration.
- Estimators provide a safe distributed training loop, it warns you about the errors at the time of building the graph, initializing variables, loading data, etc.
- Easier customization.

Presented the type of neural network chosen for this DFP, next we are going to present the parameters that have not been modified in all the executions:

- Hidden units: The hidden units are the number of hidden units per layer, also called neurons. The number of hidden units used is 100, this decision has been made after look up others Estimators where the number of possible columns in the CSV where between 2 and 5 and the number of hidden units where between 10 and 15, basically what has been done here is scale it proximately for 22 columns.

- Weight column: Used to down weight or boost examples during training. As we do not know a perfect knowledge about the matter, we have decided to keep the weights as they are.

- Optimizer: An optimization algorithm used to produce better and faster results. The chosen algorithm is Adagrad, Adagrad is an algorithm for gradient-based optimization that does just this, adapts the learning rate to the parameters, performing smaller updates [63], the decision behind using this algorithm is that eliminates the need to manually tune the learning rate, the cons of using it is that the more that is used the less that will be able to learn, this is a problem with humongous amounts of data sets, but this is not our case.

- Activation function: Activation function applied to each layer, used to determine the output, the main reason behind this decision is that the ReLU reduces the vanishing gradient problem, which consists in the reduction of the capability of the weights to be modified and therefore it reduces the capability to learn.

- Loss reduction: Describes how to reduce the training loss over batch, for this the method used is the MEAN, mean of the scalar sum of weighted losses, this method has been chosen because its simplicity.

- Batch Norm: Whether to use batch normalization after each hidden layer, we have decided to not make any normalization about the batch as it works better with complex and in a matrix form data sets.

- Feature columns: The columns inside the training set and validation set, there is no decision in this case as we have to use all the present columns.

The following parameters are the ones that we will change on different executions with the intention of obtaining better results. For a better understanding first we will present the parameters and after that we will show the different used values with its results.

- Batch size: Number of training examples used in one iteration. The number of batch sizes used are $32, 64, 100, 150, 300, 450, 500, 700, 1000, 1250, 2000, 2800$. This decision depends on the sizes of the training sets, so this will not apply to each data set distribution in Table 3.7.

- Train steps: Number of times that will do the train with the given batches of data, the number of train steps on each execution are the following $1000, 2000, 3000, 4000, 5000$ $, 6000, 7000, 8000, 9000, 10000$, this values are common for all the execution, the intention behind this variety of values is to try to see how much does it cost to train, as this is combined with the other parameters, with this obtain a wide variety of combinations.

- Number of classes: Number of elements to classify, the number of classes are defined by the number of allocation policies, as we removed Min-Min for the lack of data. Then, the number of classes that we have are the ones defined in Table 3.7.

- Number of layers: Number of hidden layers. The used number of hidden layers are: $2, 3, 4, 5$. This decision will be explained at the moment of describing the done executions.

3.6.3 Execution and Analysis

The executions will be presented in the same order that have been done and organized by the number of hidden layers they hold, as the results given by the neural network may vary, the executions with each number of steps are repeated 50 times, this way we try to avoid a lucky or an unlucky execution, the results obtained from the executions will be presented in form of mean, maximum and minimum values.

After the last execution we will compare the results obtained for the executions with the models with 2 and 3 classes, the results obtained with the executions with 5 classes will not present as we only did one execution, and the results obtained will be explained on the first execution.

The metrics used to calculate how good our neural networks are the following:

- Accuracy: Proportion of correct classifications.

$$ACC = \frac{TP + TN}{P + N}$$

  Where ACC means Accuracy or Precision, TP is the True Positive, a correct identification of a case, TN us a True negative, an incorrect classification of a case, P is the number of real Positives cases in the data and N is the number of real Negatives cases in the data.

- Loss: Loss functions represents the inaccuracy of predictions in classification problems, the higher the loss the higher the possibility of having inaccurate predictions.

$$J(f(x);y) = \sum_{i=1}^{M}(y_i - f(x_i))^2$$

Where $J(f(x);y)$ is the loss function applied to the model function for a given a set of inputs (independent variables) x and a set outputs (dependent variables) y, then, the formula states that, for each output predicted by the model, we determine how far away the prediction is from the actual value y_i Each of the M individual distances are then squared and added to give a single number indicating how well (or badly) the model function captures the structure of the data across all the datapoints [64].

- Precision or Positive Predictive Value: Fraction of relevant instances among the retrieved instances.

$$PPV = \frac{TP}{TP + FP}$$

Where PPV means Positive Predictive Value or Precision, TP is the True Positive, a correct identification of a case, FP is a False Positive, an assertion of something that is absent.

- Recall or True Positive Rate: Fraction of relevant instances that have been retrieved over the total amount of relevant instances.

$$TPR = \frac{TP}{P}$$

Where TPR means True Positive Rate or Recall, TP is the True Positive, a correct identification of a case, and P is the number of real Positives in the data.

The accuracy and the loss are present in all the executions independently of the number of classes that the execution holds. precision and recall are only available on executions with 2 number of classes, the reason behind is that Tensorflow only gives this metrics when the number of classes is 2, as we use the Tensorflow in a high level, we cannot calculate this parameters by hand.

3.6.3.1 Specific Configuration for Execution 1

The first execution is composed by 3 hidden layers and the configurations present on the Table 3.8, on this table we can see white and red rows, the white rows means that the results obtained from the execution with that specific configuration were above or in the

average, the average has been calculated by doing the mean of the executions with the same number of classes and can be seen in Table 3.9.

The red rows means that the results obtained form the execution with that specific configuration were below the average mean accuracy, therefore, this results will not be present in this sections.

With this decision we try to focus on the configurations with better results and improve them.

The last row holds the configuration for the model with 5 classes, its results will only be present on the first execution as they were to lousy. The reason behind this decision is only to remark how significant is having an enough amount of data sets for the training and validation.

Classes	Sizes			Accuracy
	Training Set	Validation Set	Batch	Mean
2	3200	800	32	0.6420
2	3200	800	500	0.7105
2	3200	800	1000	0.6727
2	3200	800	1250	0.6716
2	3200	800	2000	0.7083
2	3200	800	2800	0.6754
2	1200	300	32	0.7059
2	1200	300	300	0.6668
2	1200	300	500	0.6585
2	1200	300	700	0.6616
2	1200	300	1000	0.6966
2	560	140	32	0.7095
2	560	140	150	0.7120
2	560	140	300	0.6650
2	560	140	450	0.6648
3	50	30	32	0.3865
3	100	30	32	0.4395
3	100	30	64	0.4321
3	100	30	100	0.4388
3	150	40	32	0.4076
3	150	40	64	0.4035
3	150	40	100	0.3842
3	150	40	150	0.3932
5	50	5	32	0.1946

Table 3.8: Configuration for Execution 1

Classes	Average Accuracy
2	0.6814
3	0.4106

Table 3.9: Average Accuracy for classes 2 and 3

Execution 1, 2 Classes 3 Hidden Layers

In the following Tables 3.10, 3.11 we can see the accuracy, loss, precision and recall for the first execution with 2 classes. The accuracy column shows that the mean is around the 70% with the highest value at 75.62% and the lowest at 55.91%, to be considered good the accuracy of a neural network has to be around 95%. Also, we can see that the gap between the different results at the mean column is not wide, this is a problem, as does not tell us how to improve the model in terms of test set size and batch size.

The loss in this case gives us more information about what configuration give us a more consistent model. If we look at the mean column, we can see that the values are quite similar between them, but, we have to remember that for the loss calculation we are using the MEAN method which is the mean of the sums over the batch, then, in this case, the lower the batch the more consistent the model, in terms of loss, the models with short batches give more consistency than those with larger ones.

In terms of precision, the model obtained with this configurations is quite accurate with the results obtained in the accuracy as they follow similar percentages in the average.

The worst results obtained are easily seen in the recall column, where the values hardly reach the 50% on the average, with this, we can see that the NN lacks on resources to identify the True Positives or simply that the given training sets are to complex to learn from them in a generic way.

	Sizes			Accuracy			Loss		
Iterations	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	1200	300	32	0.7013	0.7228	0.6727	0.5796	0.6425	0.5531
2000	3200	800	500	0.7049	0.7329	0.6654	0.597	0.7276	0.5568
3000	1200	300	32	0.6958	0.7245	0.6494	0.5897	0.6462	0.5580
4000	560	140	32	0.7202	0.7526	0.6415	0.5474	0.6086	0.5258
5000	560	140	150	0.7181	0.7455	0.6738	0.5497	0.5916	0.5253
6000	3200	800	500	0.7126	0.7285	0.6929	0.5711	0.5889	0.5592
7000	560	140	32	0.6974	0.7562	0.5985	0.6073	0.8239	0.5317
8000	560	140	32	0.7153	0.7491	0.5591	0.5552	0.7525	0.5192
9000	1200	300	32	0.6976	0.7295	0.6460	0.5834	0.6416	0.5603
10000	3200	800	2000	0.7095	0.7260	0.6885	0.5964	0.6758	0.5680

Table 3.10: Execution 1: 2 Classes Accuracy and Loss

Iterations	Sizes			Precision			Recall		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	1200	300	32	0.7281	0.7675	0.6812	0.4604	0.5164	0.3983
2000	3200	800	500	0.7149	0.7326	0.7002	0.4817	0.4962	0.4623
3000	1200	300	32	0.7220	0.7735	0.6396	0.4724	0.5827	0.3817
4000	560	140	32	0.6981	0.8405	0.5903	0.5284	0.6507	0.3512
5000	560	140	150	0.7094	0.8160	0.5911	0.4995	0.67136	0.3792
6000	3200	800	500	0.7033	0.7201	0.6888	0.4945	0.5010	0.4843
7000	560	140	32	0.6879	0.7757	0.6009	0.5171	0.6509	0.3769
8000	560	140	32	0.6969	0.8068	0.5793	0.5197	0.6759	0.3758
9000	1200	300	32	0.7393	0.8	0.6330	0.4538	0.5909	0.3217
10000	3200	800	2000	0.6957	0.7248	0.6690	0.4839	0.4928	0.4772

Table 3.11: Execution 1: 2 Classes Precission and Recall

Execution 1, 3 Classes 3 Hidden Layers

The following Table 3.12, presents the results obtained from the first execution with 3 classes. Taking a look at the accuracy column we can see that the values hardly surpasses the 45% at the average. Like in the previous Table 3.10, the gaps between the different results are too narrow to tell us which configuration follow to improve the model.

A shallow look at the loss column gives us an idea of how inconsistent the model is with their respective configuration. This makes perfect logic with the accuracy obtained. On the other hand, if we look at the rows with 7000, 8000, 9000 and 10000 iterations in the average is lower than the ones with less iterations,in this case looks like higher iterations produce more consistent models.

Iterations	Sizes			Accuracy			Loss		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	100	30	32	0.4424	0.5280	0.3258	4.3787	5.9149	3.2881
2000	100	30	100	0.4539	0.5280	0.3707	2.3200	3.7001	1.4525
3000	100	30	32	0.4413	0.5505	0.3483	4.4342	5.5714	3.3387
4000	100	30	32	0.4408	0.5617	0.3370	4.1255	5.4797	3.2262
5000	100	30	100	0.4337	0.5280	0.3146	4.6024	6.4039	3.4442
6000	100	30	32	0.4424	0.5393	0.3483	3.9770	5.3232	2.9110
7000	100	30	32	0.4613	0.5617	0.3820	1.1748	1.7258	0.9727
8000	100	30	100	0.4494	0.5393	0.2584	1.4679	2.3146	1.0344
9000	100	30	64	0.4586	0.5617	0.3595	1.3230	1.9176	1.0128
10000	100	30	32	0.4469	0.5730	0.3370	1.8417	2.8374	1.0894

Table 3.12: Execution 1: 3 Classes Accuracy and Loss

Execution 1, 5 Classes 3 Hidden Layers

The results gathered for the last execution can be seen in the Table 3.13,as has been said before, the model with 5 classes are only present to remark the importance of the sets. The mean on the accuracy column can not reach the 25% at the average, this, followed with high loss values give us a totally impractical model.

Iterations	Sizes			Accuracy			Loss		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	50	5	32	0.1966	0.375	0.0416	5.7745	8.5851	1.9922
2000	50	5	32	0.1853	0.2912	0.0416	8.2448	11.0378	6.0558
3000	50	5	32	0.1883	0.3333	0.0416	7.7226	9.9075	5.7638
4000	50	5	32	0.1766	0.2916	0.0416	4.4769	6.3977	2.5369
5000	50	5	32	0.2108	0.4583	0.0833	7.0985	10.1006	3.7104
6000	50	5	32	0.1908	0.3333	0.0416	7.3009	9.6836	5.1129
7000	50	5	32	0.1941	0.3333	0.0416	6.6228	8.9838	3.8799
8000	50	5	32	0.1891	0.3333	0.0416	6.6228	8.9835	3.8799
9000	50	5	32	0.2091	0.4166	0.0416	8.4244	12.7566	5.9522
10000	50	5	32	0.2058	0.4583	0.0833	7.7428	10.5962	5.7161

Table 3.13: Execution 1: 5 Classes Accuracy and Loss

3.6.3.2 Specific Configuration for Execution 2

The second execution is composed by 4 hidden layers and the configurations present on Table 3.14, from now on the configuration will be the same for the rest of executions. As mentioned before, this configurations are the ones where the results where over the average, in the following executions we will try to improve this results.

Classes	Training Set Size	Validation Set Size	Batch Size
2	3200	800	2000
2	3200	800	500
2	1200	300	32
2	1200	300	1000
2	560	140	32
2	560	140	150
3	100	30	32
3	100	30	64
3	100	30	100

Table 3.14: Configuration for Execution 2, 3 and 4

Execution 2, 2 Classes 4 Hidden Layers

The following set of results obtained from the model with 2 classes are present in the Tables 3.15 3.16. The obtained accuracy does not improve compared with the one obtained in Execution 1 in terms of mean, maximum value and minimum values, at first we can see that the gathered values does not show any variance in terms of mean, maximum and minimum value.

The loss follows the same pattern as the accuracy, the values does not show any difference from the ones obtained in the previous execution with the same number of elements in the class with the exception of the first row, which has increased twice in comparison with the same vale in the previous execution.

The precision and the recall follows the same path as the accuracy and the loss, the values show an almost null improvement. With this execution we can see that a subtle change as it is the increment of a hidden layer does not grant us any improvement over the model with the used configuration.

Iterations	Sizes			Accuracy			Loss		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	560	140	150	0.7027	0.7278	0.6611	1.1254	2.3417	0.55507
2000	3200	800	2000	0.7070	0.7310	0.6510	0.6976	1.5855	0.5577
3000	3200	800	2000	0.6928	0.7229	0.6372	0.7611	1.7194	0.5575
4000	560	140	32	0.7107	0.7526	0.4946	0.5556	0.6932	0.5225
5000	560	140	150	0.7174	0.7526	0.6810	0.5508	0.5863	0.5193
6000	3200	800	500	0.7041	0.7285	0.6741	0.5812	0.6219	0.5578
7000	560	140	32	0.7022	0.7634	0.6236	0.6079	0.8934	0.5266
8000	560	140	32	0.7167	0.7455	0.6774	0.5517	0.6049	0.5301
9000	1200	300	32	0.6998	0.7195	0.6410	0.6774	1.0058	0.5286
10000	3200	800	2000	0.7116	0.7285	0.6754	0.5915	0.7482	0.5680

Table 3.15: Execution 2: 2 Classes Accuracy and Loss

Iterations	Sizes			Precision			Recall		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	1200	300	32	0.7275	0.7892	0.6764	0.4681	0.5360	0.3893
2000	3200	800	500	0.7159	0.7349	0.6879	0.4637	0.4903	0.4037
3000	1200	300	32	0.7173	0.7867	0.6515	0.4729	0.5438	0.4006
4000	560	140	32	0.7027	0.8131	0.5877	0.5230	0.6604	0.4057
5000	560	140	150	0.7174	0.825	0.6220	0.4989	0.6095	0.3792
6000	3200	800	500	0.7005	0.7235	0.6687	0.4969	0.5407	0.4856
7000	560	140	32	0.6976	0.7865	0.5891	0.5108	0.6373	0.3808
8000	560	140	32	0.6999	0.8481	0.5338	0.5234	0.7623	0.3730
9000	1200	300	32	0.7402	0.7935	0.6613	0.4523	0.5622	0.3634
10000	3200	800	2000	0.6987	0.7170	0.6784	0.4822	0.4933	0.4742

Table 3.16: Execution 2: 2 Classes Precision and Recall

Execution 2, 3 Classes 4 Hidden Layers

The values retrieved through this execution are present in the Table 3.17. As in the model with 2 classes execution, the values present in the accuracy column as well as the values in the loss column does not show any improvement with the exception of the values of the rows with 7000 and 10000 iterations where such values are almost 4 times higher than in the first execution with 3 classes.

	Sizes			Accuracy			Loss		
Iterations	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	100	30	32	0.4505	0.5730	0.3707	4.6547	5.764	2.7576
2000	100	30	32	0.4512	0.5280	0.3258	3.5568	5.7484	1.2331
3000	100	30	32	0.4521	0.5393	0.3595	4.8050	6.2888	3.7536
4000	100	30	64	0.4469	0.5617	0.3370	4.0560	6.3557	2.3495
5000	100	30	64	0.4453	0.5280	0.3595	5.3317	7.0458	3.8275
6000	100	30	100	0.4480	0.5280	0.325	5.2265	7.6972	4.0027
7000	100	30	64	0.4370	0.5280	0.3258	4.8238	5.9901	3.8080
8000	100	30	100	0.4579	0.5730	0.337	1.3826	2.4203	0.9776
9000	100	30	64	0.4550	0.5393	0.3707	1.2778	1.9861	0.9806
10000	100	30	100	0.4539	0.5842	0.3370	4.5464	6.5645	3.3230

Table 3.17: Execution 2: 3 Classes Accuracy and Loss

3.6.3.3 Specific Configuration for Execution 3

The third execution is composed of 5 hidden layers, the rest of the configuration values remains the same as in the previous execution.

Execution 3, 2 Classes 5 Hidden Layers

The following Tables 3.18 3.19 hold the results obtained from the third execution using the configurations for the model with 2 classes.

The first Table 3.18 shows an accuracy almost identical to the ones obtained previously in the model with 2 classes, if we look closely we can see an slightly increase of the accuracy of 1% this low increment cannot be counted as a real improvement as it is not high enough to not be considered lucky, also is not present in all the iterations.

If we look at the loss, we can see a decrease in comparison with the other executions with the same classes, with the exception of the 1000 iterations row at the second execution.

In terms of precision the mean values have decreased in almost all the iterations compared with the the previous execution. On the other hand the recall we have not get any increase nor decrease of the values.

	Sizes			Accuracy			Loss		
Iterations	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	3200	800	2000	0.7125	0.7335	0.6541	0.6896	1.6021	0.5460
2000	3200	800	2000	0.7133	0.7285	0.6710	0.6146	0.9440	0.5538
3000	3200	800	2000	0.6997	0.7329	0.6566	0.6965	1.1646	0.5427
4000	560	140	32	0.7227	0.7562	0.6523	0.5453	0.5854	0.5208
5000	560	140	150	0.7125	0.7491	0.6308	0.5555	0.6999	0.5126
6000	560	140	32	0.7124	0.7526	0.6344	0.5587	0.63393	0.5168
7000	560	140	32	0.6891	0.7491	0.4982	0.6077	0.7803	0.5266
8000	560	140	32	0.7128	0.7455	0.6379	0.5519	0.6163	0.5170
9000	1200	300	32	0.7042	0.7295	0.6477	0.5798	0.6354	0.5642
10000	3200	800	2000	0.7149	0.7292	0.6991	0.5787	0.6222	0.5655

Table 3.18: Execution 3: 2 Classes Accuracy and Loss

Iterations	Sizes			Precision			Recall		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	560	140	150	0.6468	0.6978	0.5762	0.5260	0.6251	0.2612
2000	3200	800	2000	0.7047	0.7359	0.6457	0.4775	0.4904	0.4525
3000	3200	800	2000	0.6655	0.7044	0.6261	0.5283	0.5419	0.5091
4000	560	140	32	0.6854	0.8170	0.0	0.5267	0.6414	0.3709
5000	560	140	150	0.7212	0.8271	0.6354	0.4961	0.6104	0.3821
6000	3200	800	500	0.6871	0.7202	0.6424	0.5080	0.5970	0.4876
7000	560	140	32	0.6891	0.7727	0.5751	0.5237	0.7013	0.4015
8000	560	140	32	0.7034	0.8378	0.6226	0.5255	0.6378	0.3913
9000	1200	300	32	0.7452	0.7860	0.6796	0.4464	0.5127	0.3740
10000	3200	800	2000	0.7014	0.7292	0.6616	0.4802	0.4913	0.4433

Table 3.19: Execution 3: 2 Classes Precision and Recall

Execution 3, 3 Classes 5 Hidden Layers

The present Table 3.20 holds the results obtained for the third execution with 2 classes, in terms of accuracy we can an slightly improvement of it on the average compared with the first execution while keeping identical results with the second execution.

In terms of loss the values are quite similar, with slight differences in some values compared with both executions, not improving nor diminish the performance of the model.

Iterations	Sizes			Accuracy			Loss		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	100	30	32	0.4577	0.5393	0.3370	4.7008	6.0318	2.9625
2000	100	30	100	0.4480	0.5280	0.359	2.3662	4.0905	1.0329
3000	100	30	100	0.4496	0.5617	0.303	5.6577	8.0034	4.1382
4000	100	30	32	0.4530	0.5617	0.3820	4.8108	6.2757	3.4073
5000	100	30	64	0.4532	0.5280	0.3370	5.5543	8.0106	4.0742
6000	100	30	64	0.4579	0.5617	0.3707	3.5759	5.6810	1.5422
7000	100	30	32	0.4728	0.5505	0.3258	1.0685	1.3718	0.9573
8000	100	30	32	0.4541	0.5393	0.3707	5.0275	6.5362	3.7382
9000	100	30	64	0.4582	0.5393	0.3258	1.2956	2.2319	0.9700
10000	100	30	64	0.4523	0.5505	0.3595	4.9325	6.1755	3.5949

Table 3.20: Execution 3: 3 Classes Accuracy and Loss

3.6.3.4 Specific Configuration for Execution 4

The fourth execution is composed by 2 hidden layers, in difference of the previous executions in this one we have decreased the number of hidden layers, the reason behind is, after notice that adding more hidden layers has almost no effect on the results, we wanted to see if the removing some of them would make a difference.

Execution 4, 2 Classes 2 Hidden Layers

The results gathered for the fourth execution with 2 classes are present in Tables 3.21 3.22. In the first one, we can see an accuracy similar to the other executions, the mean values are still around $70 - 71\%$, at first we cannot say that the number of layers does not make any influence over the model, but we can say that in accuracy terms the difference is not decisive in this case.

The loss in this case we can see is quite low and stable, in relation to the values obtained in the accuracy, also in the max column we can see that the values only surpass the 71% in one occasion while in the rest is quite close to the mean values so we could label that case as an unfortunate execution.

For the precision values we can see that the achieved precision is quite high for the previously seen values, we can also see that in the mean column the values are do not differ to much between them, but, they do in some cases with the values in the mean, max and min columns, as an example we can see the iterations rows 4000 or 5000, in those cases we con ensure that the mean defines the reality as it could just be the mean of values more close to the minimums and maximum values.

The recall on the other hand does not go as well as the other metrics, even though if we take a look at the min and max values we have as before a huge gap that does not ensure us that the mean values are really what we have at the table.

Iterations	Sizes			Accuracy			Loss		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	3200	800	500	0.6981	0.7160	0.6747	0.5848	0.62271	0.5578
2000	3200	800	500	0.7119	0.7304	0.6948	0.5741	0.6002	0.5516
3000	1200	300	32	0.6986	0.7212	0.6560	0.5838	0.6273	0.5524
4000	560	140	32	0.7150	0.745	0.645	0.6012	0.6642	0.5701
5000	560	140	150	0.7108	0.74193	0.6451	0.5584	0.6487	0.5223
6000	3200	800	500	0.7135	0.7285	0.6916	0.5684	0.5834	0.5529
7000	3200	800	500	0.7036	0.7254	0.6710	0.6060	0.7106	0.5407
8000	560	140	32	0.711	0.756	0.630	0.5573	0.6217	0.5165
9000	1200	300	32	0.6984	0.7212	0.6143	0.5827	0.6604	0.5626
10000	3200	800	2000	0.7054	0.7223	0.6797	0.6048	0.6933	0.5680

Table 3.21: Execution 4: 2 Classes Accuracy and Loss

Iterations	Sizes			Precision			Recall		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	1200	300	32	0.7281	0.7675	0.6812	0.4604	0.5164	0.3983
2000	3200	800	500	0.7149	0.7326	0.7002	0.4817	0.4962	0.4623
3000	1200	300	32	0.7220	0.7735	0.6396	0.4724	0.5827	0.3817
4000	560	140	32	0.6981	0.8405	0.5903	0.5284	0.6507	0.3512
5000	560	140	150	0.7094	0.8160	0.5911	0.4995	0.67136	0.3792
6000	3200	800	500	0.7033	0.7201	0.6888	0.4945	0.5010	0.4843
7000	560	140	32	0.6879	0.7757	0.6009	0.5171	0.6509	0.3769
8000	560	140	32	0.6969	0.8068	0.5793	0.5197	0.6759	0.3758
9000	1200	300	32	0.7393	0.8	0.6330	0.4538	0.5909	0.3217
10000	3200	800	2000	0.6957	0.7248	0.6690	0.4839	0.4928	0.4772

Table 3.22: Execution 4: 2 Classes Precision and Recall

Execution 4, 3 Classes 2 Hidden Layers

The next table hold the values retrieved from the fourth execution with 3 classes. In terms of accuracy we can see the highest values at the 2000 iterations and from $7000 - 10000$ reaching almost the 46% in all the cases but one that surpasses the percentage.

For the loss we can see pretty low values specially for the latest iterations, $7000 - 10000$, the rest for the execution follow the usual loss for the results obtained, similar to the ones previously seen.

Iterations	Sizes			Accuracy			Loss		
	Training Set	Validation Set	Batch	Mean	Max	Min	Mean	Max	Min
1000	100	30	32	0.4467	0.5168	0.3595	3.3830	4.5902	2.4308
2000	100	30	100	0.4593	0.5505	0.3485	1.7549	2.7110	1.1304
3000	100	30	32	0.4408	0.5168	0.3258	3.6112	4.8324	2.5624
4000	100	30	100	0.4469	0.5505	0.3707	3.1077	4.2130	2.0854
5000	100	30	32	0.4433	0.5730	0.3033	3.5650	4.8531	1.8661
6000	100	30	32	0.4384	0.5505	0.3707	2.8602	3.8632	1.8359
7000	100	30	32	0.4584	0.5617	0.3595	1.1273	1.4368	0.9381
8000	100	30	100	0.4588	0.5617	0.3595	1.2384	1.7328	0.9877
9000	100	30	64	0.4667	0.5505	0.2921	1.1730	1.6129	0.9832
10000	100	30	32	0.4588	0.5730	0.3595	1.3993	1.8941	1.0480

Table 3.23: Execution 4: 3 Classes Accuracy

3.6.3.5 Overall Results Analysis

In the following charts only the mean columns will be shown, even though the maximum and minimum value give us an idea on how much the values can fluctuate, can also trick us as they can be product of a lucky and/or unlucky execution, to avoid this we are going to center us in the mean, which is the result of 50 executions, this way we can prevent to the extend possible those fortunate and/or unfortunate executions.

2 classes The first Figure 3.9 shows the means of the accuracy with its respectively number of iterations of all the executions for the models with 2 classes. For the iterations 1000, 2000 and 3000 and 4000, after all this iterations its accuracy goes down until the iteration 9000 and 10000 which becomes the best model in terms of accuracy. The second best model in terms of accuracy would be the one in the fourth execution which has its peaks in the iterations 6000 and 7000, the models used at the first and second execution having its peaks at the iterations 5000 and 8000 respectively.

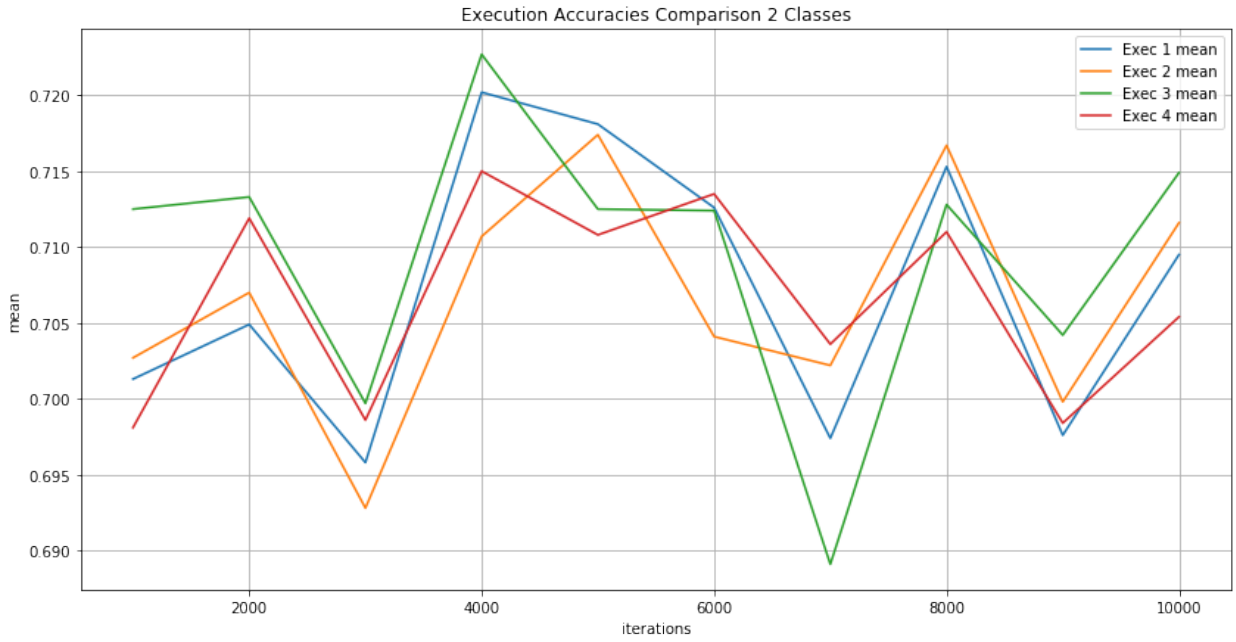

Figure 3.9: Chart with the means of the all executions for the models with 2 classes[65].

In terms of loss present in the Figure 3.10 the bests models seems to be the forth and the third execution, the fourth model gets the best loss results with the iterations 1000, 2000 and 3000, while in the rest of the iterations loss results obtained at the others execution are really close, even though the third execution seems to be insignificantly more consistent. For the first execution we can say that is the most inconsistent.

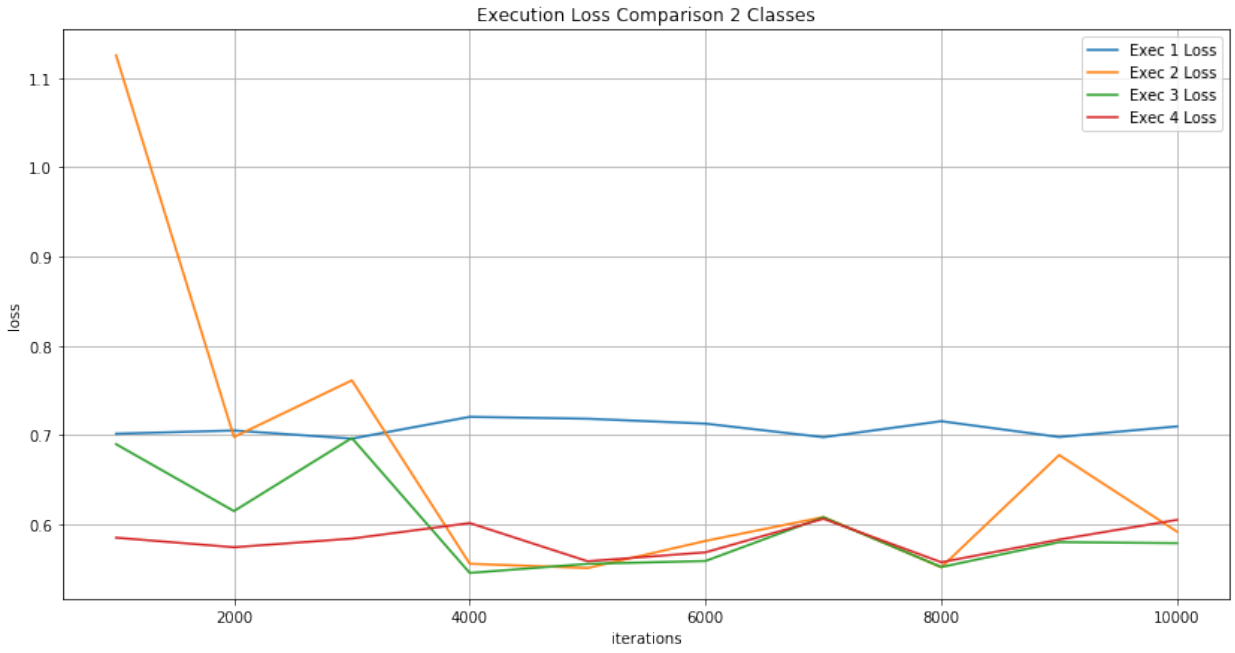

Figure 3.10: Chart with the loss of the all executions for the models with 2 classes[66].

In precision terms present in the Figure 3.11 we can see that the first and the fourth execution models hold the best precision values for the iterations, 1000, 2000, 3000 and 4000. The rest of the iterations are controlled by the first, second and fourth executions. This time the third execution gets the worst results.

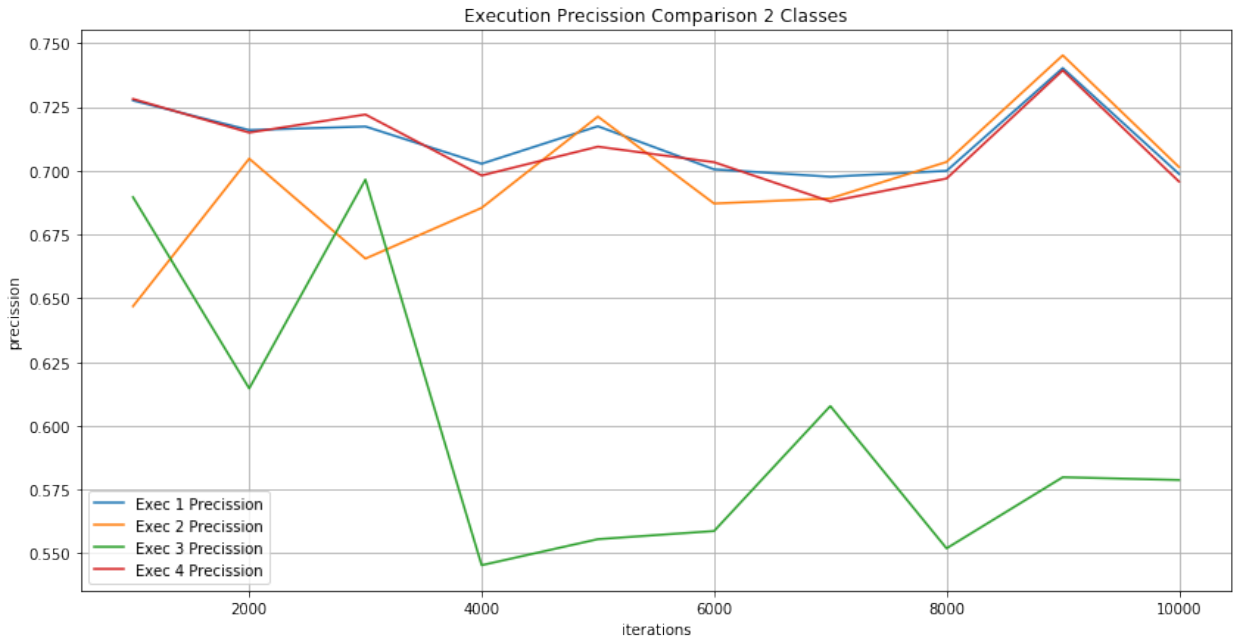

Figure 3.11: Chart with the Precision of the all executions for the models with 2 classes[67].

Finally for the executions with 2 classes we have the recall, which results are present in Figure 3.12 we can see clearly that the best in terms of recall is the third execution which values decrease over the increase of the iterations. The rest of the executions are quite close starting from the 4000 iteration to the 10000 iteration, initially the second best results would be the obtained with the second execution with awful values.

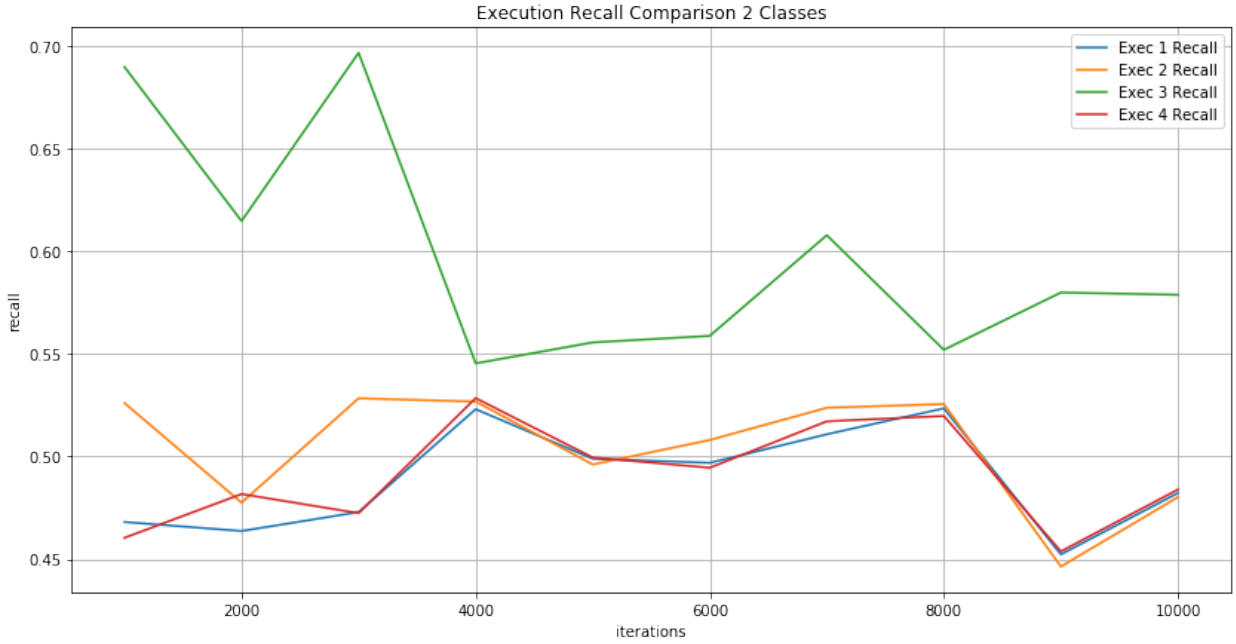

Figure 3.12: Chart with the recall of the all executions for the models with 2 classes[68].

3 classes The Figure 3.13 shows the means of accuracy of all the executions with 3 classes. In this graph we can see that the values are more dispersed compared with the values in the accuracy of the 2 classes. The first important thing to remark is that it cannot even reach the 50%. Secondly we can see that the third executions stands out over the others even though as we have said before, the results are awful. Curiously the second best model is the one obtained in the fourth execution with 2 hidden layers, standing out in iterations with a huge gap between them, iterations 2000 and 8000 to 10000. The worst model is clearly the one of the first execution, which also hold the lowest values in the execution.

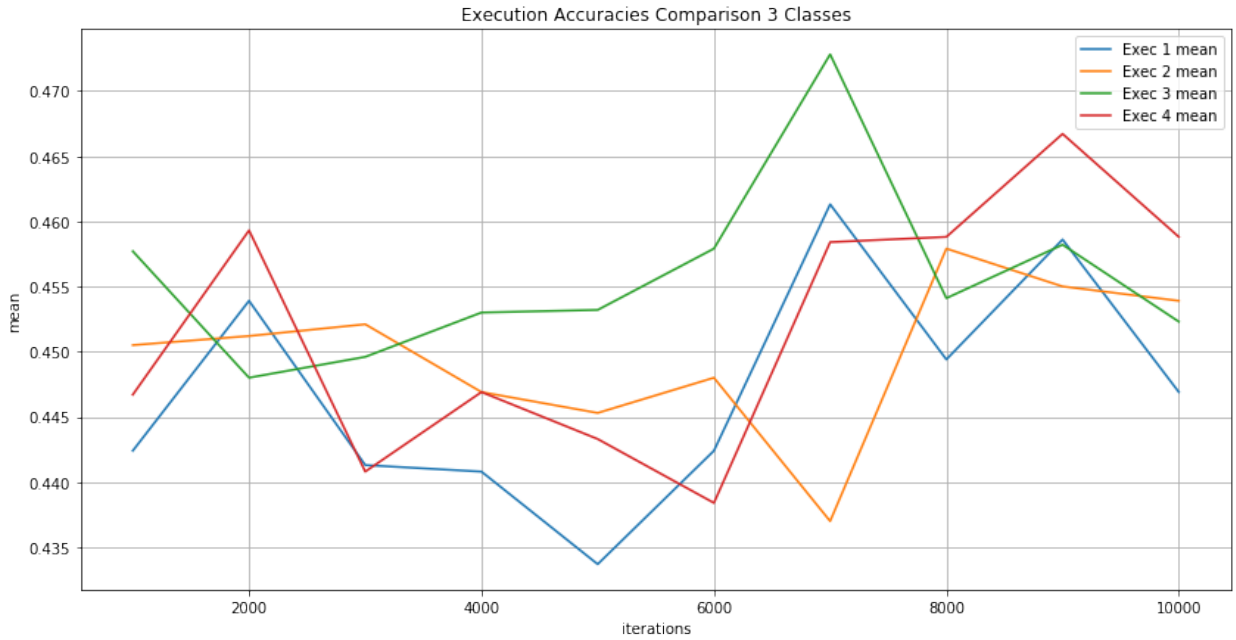

Figure 3.13: Chart with the means of the all executions for the models with 3 classes[69].

The Figure 3.14 holds the loss values for the executions with 3 classes. As in the previous Figure 3.13 where the third execution held the majority of the best results, in this cases happens to hold the worst of them with the exception of the iteration 7000 which is almost the same for all the executions. On the other hand the last execution holds the best values for the loss with, watching at the Figure 3.10 we can see that in this case, less hidden layers give us best loss results.

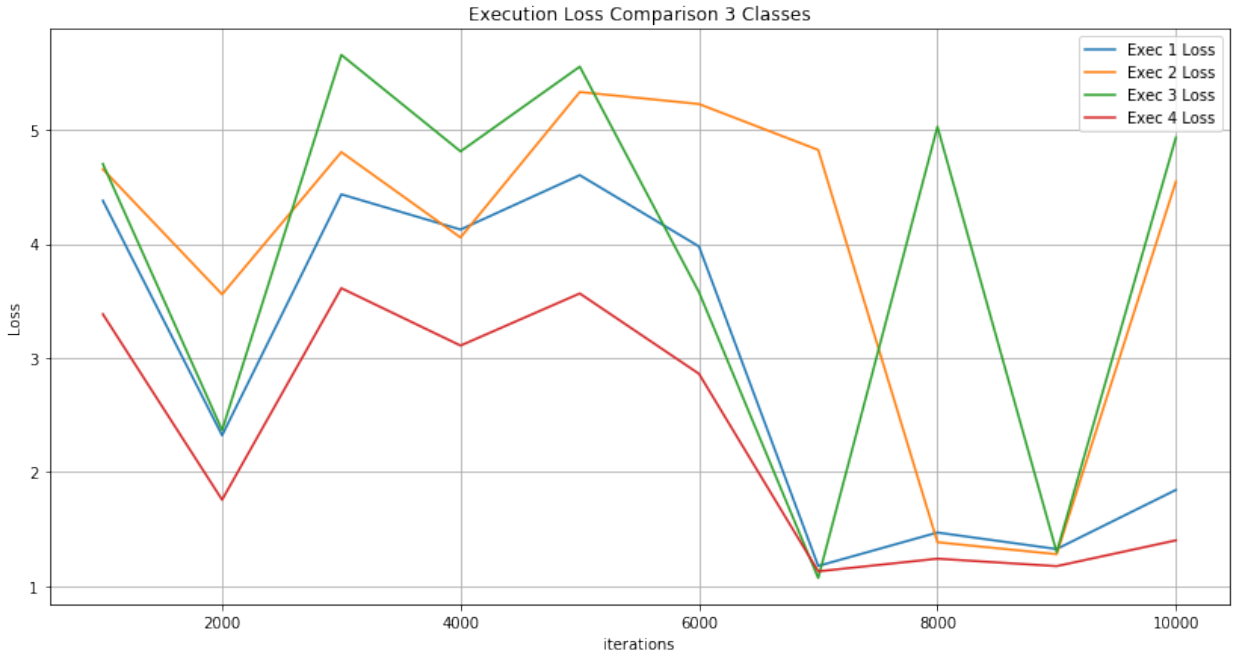

Figure 3.14: Chart with the loss of the all executions for the models with 3 classes[70].

Chapter 4

General Conclusions and Future Work

In this chapter, the research work presented in this DFP is summarized. It also gives some suggestions that might make possible the acquirement of favourable results that satisfy our necessities.

4.1 Conclusion

In this DFP we have proposed the possibility of using neural networks to find which is the best heuristic to allocate the applications arriving to the system, obtaining the best results in terms of time and energy consumption. Our proposal focus on optimizing the execution time, even though the methodology could be easily adapted to try to optimize the energy consumption.

The test sets were obtained by executing different real traces of workloads extracted from the ParallelWorkload Archive Site created by Feitelson in [71] GridSim, while the neural network has been implemented using Estimators from Tensorflow. The obtained results were too lousy to be considered possible the usage of such techniques for our purpose, with this we are not corroborating that the application of neural networks for this matter is impossible, only that our method is not efficient to achieve our intentions.

Firstly we have to take into account the limitations produced by the number of data sets, there could be the case that increasing this number we could get better results.

Secondly we have chosen an specific representation, there could be other representations which could make the learning process easier than the one we have picked, this decision goes along with the kind of neural network used for the implementation, usually specific representations tend to work better with specific kinds of neural networks, this

also is related to the kind of problem.

Finally but not less important the decisions made about the parameters are also important facts to take into account, sometimes little differences while tuning a NN can make a huge impact, therefore it is important to keep in mind that every single decision we make in a NN, can change significantly the final results.

With this, the only statement we can do about this DFP is that the specific methodology that we have developed and followed it is not the proper one to satisfy the necessities that our problem requires.

4.2 Future Work

Previously we have remarked some limitations and some aspects to take into account when making a neural network, as a future work we would like to suggest some corrections that might produce better results.

Larger Amounts of Data and Different Tuning

Repeat the process with larger amounts of data, around 10000 sets for each method if possible, also study the possibility of tuning and configuring the NN in a different way, also, try different proportion of training sets and validation sets .

Different Representation and/or Kind of Neural Network

Change the representation, another representation might make easier the learning process. Depending on the representation there might be the need of changing also the kind of neural network. Also exist the possibility of just changing the kind of neural network.

Using a Different Framework

Even though Tensorflow is one of the bests frameworks, there are others with similar characteristics, maybe some of them are easy to use, therefore, making this task easier.

Bibliography

[5] Eloy Gabaldon Ponsa. *Meta-Heuristics for Scheduling in Cluster Federated Environments*. 2018, p. 31.

[6] Eloy Gabaldon Ponsa. *Meta-Heuristics for Scheduling in Cluster Federated Environments*. 2018, pp. 32–33.

[7] Eloy Gabaldon Ponsa. *Meta-Heuristics for Scheduling in Cluster Federated Environments*. 2018, pp. 33–34.

[8] Priti Lohani. *Cluster Scheduling*. 2009, pp. 5–9.

[9] Eloy Gabaldon Ponsa. *Meta-Heuristics for Scheduling in Cluster Federated Environments*. 2018, pp. 35–37, 40–43.

[10] Josep Lluís Lérida Monsó. *Meta-Planificador Predictivo para Entornos Multicluster no Dedicados. PhD thesis, Universitat Autonoma de Barcelona*. 2009.

[11] Héctor Blanco de Frutos. *Clusterización de aplicaciones paralelas para su planificación en entornos de cómputo multi-cluster. PhD thesis, Universitat de Lleida*. 2012.

[12] Eloy Gabaldon Ponsa. *Meta-Heuristics for Scheduling in Cluster Federated Environments*. 2018, pp. 40–43.

[13] Laurent Lefevre Anne-Cécile Orgerie and Jean-Patrick Gelas. *Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems*. 2008, pages 171–178.

[15] Gavin Hackeling. *Mastering Machine Learning with scikit-learn*. 2016, pp. 7–11.

[19] Ramón Béjar Torres. *Inteligencia Artificial: Aprendizaje automático*. 2008, pp. 26–29.

[26] Ramón Béjar Torres. *Inteligencia Artificial: Aprendizaje automático*. 2008, pp. 32–37.

[27] Dror G. Feitelson and Larry Rudolph. *Distributed hierarchical control for parallel processing*. 1990, pp. 65–77.

[28] Thyagaraj Thanalapati and Sivarama Dandamudi. *An efficient adaptive scheduling scheme for distributed memory multicomputers*. 2001, pp. 758–768.

[29] Victor M. Albornoz. *Multiple job co-allocation strategy for heterogeneous multi-cluster systems based on linear programming. The Journal of Supercomputing*. 2011, pp. 394–402.

[30] Victor M. Albornoz. *MIP model scheduling for BSP parallel applications on multicluster environments*. 2012, pp. 12–18.

[31] Albert Y. Zomaya and Yee-Hwei Teh. *Observations on using genetic algorithms for dynamic load-balancing*. 2001, pp. 899–911.

[32] Richard F. Freund. *A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems*. 2001, pp. 810–837.

[33] Fatos Xhafa Javier Carretero and Ajith Abraham. *Genetic algorithm based schedulers for grid computing systems*. 2007, pp. 1–19.

[34] K Thanushkodi and K Deeba. *A new improved particle swarm optimization algorithm for multiprocessor job scheduling*. 2011.

[35] Anil Kumar Sharma Surendra Kumar Patel and Anurag Seetha. *Implementing job scheduling to optimize computational tasks in grid computing using pso*. 2015, pp. 20–24.

[36] José Ranilla Alberto Cocaña and Luciano Sánchez. *Energy-efficient allocation of computing node slots in HPC clusters through parameter learning and hybrid genetic fuzzy system modeling*. 2015, pp. 1163–1174.

[37] Ronald P. Doyle. Managing energy and server resources in hosting centers. *ACM SIGOPS Operating Systems Review*. 2001, pp. 103–116.

[38] S Tamil Selvi M Christobel and Shajulin Benedict. *Efficient scheduling of scientific workflows with energy reduction using novel discrete particle warm optimization and dynamic voltage scaling for computational grids*. 2015.

[39] Albert Y. Zomaya. *Energy efficient genetic-based schedulers in computational grids*. 2015, pp. 809–829.

[40] Rajkumar Buyya Kyong Hoon Kim and Jong Kim. *Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters*. 2007, pp. 541–548.

[41] Eloy Gabaldon Ponsa. *Meta-Heuristics for Scheduling in Cluster Federated Environments*. 2018, pp. 46–48.

[42] *Multi-criteria genetic algorithm applied to scheduling in multi-cluster environments*. 2015, pp. 287–295.

[43] *Particle Swarm Optimization scheduling for Energy Saving in Cluster Computing Heterogenous Environments*. 2016.

[44] *Blacklist Multi-objective Genetic Algorithm for Energy Saving in Heterogeneous Environments*. 2017, pp. 354–369.

[45] *Energy Efficient Scheduling on Heterogeneous Federated Clusters using a Fuzzy Multi-Objective Meta-heuristic*. 2017.

[46] *Natural Language Processing: State of The Art, Current Trends and Challenges*. 2017, pp. 1, 9–11.

[50] *Energy Efficient Scheduling on Heterogeneous Federated Clusters using a Fuzzy Multi-Objective Meta-heuristic*. 2018.

[51] Daniel C. Stanzione Jr. *Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. The Journal of Supercomputing*. 2005, pp. 135–163.

[52] Jonathan Wagner. *Online resource matching for heterogeneous grid environments. In CCGRID*. 2005, pp. 607–614.

[53] Yi Liu Yu Li and Depei Qian. *A heuristic energy-aware scheduling algorithm for heterogeneous clusters. In Parallel and Distributed Systems (ICPADS)*. 2009, pp. 407–413.

[54] Jordi Planes. *Blacklist muti-objective genetic algorithm for energy saving in heterogeneous environments*. 2017, pp. 354–369.

[62] Giancarlo Zaccone. *Getting Started with TensorFlow*. 2014, pp. 7, 25–26, 35.

Webography

[1] IBM. *What is distributed computing*. URL: https://www.ibm.com/support/knowledgecenter/en/SSAL2T_8.2.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distd_comptg.html (visited on 12/13/2017).

[2] Marketing Team at ESDS Software Solution Pvt. Ltd. *Cluster Computing: Definition and Architecture of a Cluster*. March 2014. URL: https://www.esds.co.in/blog/cluster-computing-definition-and-architecture-of-a-cluster/#sthash.PGRehVWJ.dpbs (visited on 12/13/2017).

[3] Basappa B. Kodada. *Secure Communication and Computation in the Grid Cluster Computing Environment*. November 2010. URL: https://www.researchgate.net/publication/216363652_Secure_Communication_and_Computation_in_the_Grid_Cluster_Computing_Environment (visited on 12/13/2017).

[4] Matt Chew Spence Steen R. Hunt Matt Linton. *Cloud Computing Architecture, IT Security and Operational Perspectives*. August 2010. URL: https://www.nasa.gov/ppt/482833main_2010_Tuesday_5_Hunt_Linton_ChweSpence.ppt. (visited on 12/13/2017).

[14] Colin Fraser. *Machine Learning 101 supervised, unsupervised, reinforcement and beyond*. 2017. URL: https://blog.brainstation.io/machine-learning-supervised-unsupervised-reinforcement/ (visited on 01/04/2018).

[16] Mathworks Team. *What Is Deep Learning?* URL: https://uk.mathworks.com/discovery/deep-learning.html (visited on 01/10/2018).

[17] UFLDL team. *Supervised Convolutional Neural Network*. URL: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/ (visited on 01/10/2018).

[18] Christos Stergiou and Dimitrios Siganos. *Neural Networks*. URL: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Introduction%20to%20neural%20networks (visited on 01/14/2018).

[22] Michael Nielsen. *Neural Networks and Deep Learning*. Dec 2017. URL: http://neuralnetworksandde
com/chap1.html (visited on 05/13/2018).

[23] CS231N Staff. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https:
//cs231n.github.io/neural-networks-1/ (visited on 07/21/2018).

[24] University of Wisconsin Madison Staff. *A Basic Introduction To Neural Networks*. URL:
http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html (visited on
05/13/2018).

[47] Saurabh Kumar Garg. *Gridsim simulation framework*. 2001. URL: http://www.buyya.
com/gridsim/ (visited on 05/13/2018).

[49] The Rachel, Benin school of Computer Science, and Engeneering. *Logs of Real Parallel Workloads from Production Systems*. URL: http://www.cs.huji.ac.il/labs/
parallel/workload/logs.html (visited on 09/26/2017).

[63] Sebastian Ruder. *An Overview of Gradient Descent Optimization Algorithms*. 2016. URL:
http://ruder.io/optimizing-gradient-descent/index.html (visited on
08/17/2018).

[64] Dustin Stansbury. *Cutting Your Losses: Loss Functions And the Sum of Squares Loss*. 2012. URL: https://theclevermachine.wordpress.com/2012/02/13/cutting-
your-losses-loss-functions-predominance-of-sum-of-squares/ (visited on
07/18/2018).

[71] *Logs of Real Parallel Workloads from Production Systems*. URL: http://www.cs.huji.
ac.il/labs/parallel/workload/logs.html (visited on 09/26/2017).

[72] *History of Neural Networks - Stanford*. URL: https://cs.stanford.edu/people/
eroberts/courses/soco/projects/neural-networks/History/history1.html
(visited on 07/02/2018).

[73] BBC reporters team. *Google achieves AI 'breakthrough' by beating Go champion*. January 2016. URL: http://www.bbc.com/news/technology-35420579 (visited on
07/16/2018).

[74] DeepMind team. *Google DeepMind Challenge Match: Lee Sedol vs AlphaGo*. March
2016. URL: https://www.youtube.com/watch?v=vFr3K2DORc8&t=1h57m (visited
on 07/16/2018).

[75] Sam Byford. *AlphaGo beats Ke Jie again to wrap up three-part match*. May 2017. URL: https://www.theverge.com/2017/5/25/15689462/alphago-ke-jie-game-2-result-google-deepmind-china (visited on 07/23/2018).

Imageography

[20] Harsh Pokharna. *Neuron model*. Jul 2016. URL: https://medium.com/technologymadeeasy/for-dummies-the-introduction-to-neural-networks-we-all-need-c50f6012d5eb (visited on 07/21/2018).

[21] Ramón Béjar Torres. *Learning IA Book*. Aug 2008.

[25] CS231N Staff. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/neural-networks-1/ (visited on 07/21/2018).

[48] Jordi Ricard Onrubia Palacios. *Methodology Diagram*.

[55] Jordi Ricard Onrubia Palacios. *CEA-Curie Runtime/job distribution*.

[56] Jordi Ricard Onrubia Palacios. *HPC2N Runtime/job distribution*.

[57] Jordi Ricard Onrubia Palacios. *RICC Runtime/job distribution*.

[58] Jordi Ricard Onrubia Palacios. *CEA-Curie Runtime/group*.

[59] Jordi Ricard Onrubia Palacios. *HPC2N Runtime/group*.

[60] Jordi Ricard Onrubia Palacios. *RICC Runtime/group*.

[61] Jordi Ricard Onrubia Palacios. *Representation of a Single workload*.

[65] Jordi Ricard Onrubia Palacios. *Chart Means with 2 class elements*.

[66] Jordi Ricard Onrubia Palacios. *Chart Loss with 2 class elements*.

[67] Jordi Ricard Onrubia Palacios. *Chart Precision with 2 class elements*.

[68] Jordi Ricard Onrubia Palacios. *Chart Recall with 2 class elements*.

[69] Jordi Ricard Onrubia Palacios. *Chart Means with 3 class elements*.

[70] Jordi Ricard Onrubia Palacios. *Chart Loss with 3 class elements*.